ANALYTIC MODELS FOR MEMORY INTERFERENCE IN MULTIPROCESSOR COMPUTER SYSTEMS

Dileep P. Bhandarkar

Carnegie-Mellon University

Prepared for:

Air Force Office of Scientific Research
Advanced Research Projects Agency

September 1973

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR - TR - 74 - 0099	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER AD 773 843
4. TITLE (and Subtitle) ANALYTIC MODELS FOR MEMORY INTERFERENCE IN MULTIPROCESSOR COMPUTER SYSTEMS		5. TYPE OF REPORT & PERIOD COVERED Interim
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Dileep P. Bhandarkar		8. CONTRACT OR GRANT NUMBER(s) F44620-73-C-0074
9. PERFORMING ORGANIZATION NAME AND ADDRESS Carnegie-Mellon University Department of Computer Science Pittsburgh, Pennsylvania 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61101D AO 2466
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd Arlington, Virginia 22209		12. REPORT DATE September 1973
		13. NUMBER OF PAGES 187
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Air Force Office of Scientific Research (NM) 1400 Wilson Blvd Arlington, Virginia 22209		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis develops analytic models for estimating the amount of memory interference in multiprocessor systems, in which n processors access m memories independently. The processors are characterized by a typical processing time per memory access and the memories by an access time(ta) and rewrite time (tw). Processor behavior is simplified to an ordered sequence of a memory request followed by a certain amount of processing. The predominant technique used involves discrete time Markov chain models. Some simple exponential server models as well as several approximate models are also presented. Simulation

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block 20. Abstract (Continued)

is used to evaluate the accuracy of the approximate models. Some empirical measurements of the PDP-11/20 are used to estimate the parameters of a model, that is used to predict the performance of C.mmp, Carnegie-Mellon University's multiprocessor computer, which will include upto 16 PDP-11 processors.

ANALYTIC MODELS FOR MEMORY INTERFERENCE IN MULTIPROCESSOR COMPUTER SYSTEMS

Dileep P. Bhandarkar

Department of Electrical Engineering
Carnegie-Mellon University
Pittsburgh, Pa.

September 1973

Submitted to Carnegie-Mellon University
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-73-C-0074) and is monitored by the Air Force Office of Scientific Research. This document has been approved for public release and sale; its distribution is unlimited.

ABSTRACT

This thesis develops analytic models for estimating the amount of memory interference in multiprocessor systems, in which n processors access m memories independently.

The processors are characterized by a typical processing time per memory access and the memories by an access time(ta) and rewrite time(tw). Processor behavior is simplified to an ordered sequence of a memory request followed by a certain amount of processing. The predominant technique used involves discrete time Markov chain models. Some simple exponential server models as well as several approximate models are also presented. Simulation is used to evaluate the accuracy of the approximate models. Some empirical measurements of the PDP-11/20 are used to estimate the parameters of a model, that is used to predict the performance of C.mmp, Carnegie-Mellon University's multiprocessor computer, which will include upto 16 PDP-11 processors.

The models can be partitioned into three broad classes : tp=tw, tp>tw and tp<tw, where tp denotes the average processing time. Systems with tp=tw are described first because they represent boundary conditions for the other two cases. Different modeling techniques are examined for tp=tw and a reasonable

approximation is proposed. An important result observed is the absence of a law of diminishing returns. The performance of a multiprocessor system with n processors and n memories continues to rise at a constant rate as n increases. A simple exponential server model showed this rate to be *0.5*; a constant processing time model predicted a slope of *0.586* for the average number of busy Mp's. The exponential server model gives the average number of busy Mp's as $n*m/(n+m-1)$. An approximate result for constant processing times gives the average number of busy Mp's as $i*[1-(1-1/i)\uparrow j]$, where i=max(n,m) and j=min(n,m). An intuitively obvious conclusion limits the maximum number of active Pc's by min(n,m).

Markov chain models are also developed for systems with tp>tw. A new model for geometrically distributed processing time is developed. A different analytic approach is used to model systems with private caches for the processors. In general, since the Pc is slow, it takes fewer memory units for the performance to exhibit a saturation effect. In the absence of memory contention (which is now possible even for m<n) the maximum memory access rate(*MpAR*) is $n/(ta+tp)$.

With tp<tw, since the processor is fast performance improvement is obtained for m>n. If m=n, these systems do not yield significant improvement over systems with tp=tw. In general, adding an extra memory improves the performance more than adding an extra processor. The maximum average *MpAR* is the minimum of m/tc and $n/(ta+tp)$; the maximum is achieved if the processors do not interfere. Note that since the Pc is very fast, it can make a request to the memory that served it last before the rewrite cycle is over. In this case, the Pc has to wait even though no other Pc is being serviced by the memory module.

ACKNOWLEDGEMENTS

I would like to thank Professor Samuel H. Fuller for his thoughtful guidance throughout the research reported in this thesis. I am also grateful to Professor J.W. McCredie Jr. for the many discussions with me during the early stages of the research. Acknowledgement is also due to Professors Bell, Grason and Siewiorek, who helped shape the final form of the dissertation. Since this thesis was printed on the Computer Science Department's Xerox Graphics Printer, everyone involved in the XCRIBL System effort deserves thanks. Finally, a special debt is owed to Ms. Lucile O'Donnell for her help and encouragement during the course of this research.

TABLE OF CONTENTS

TABLES

FIGURES

CHAPTER 1

INTRODUCTION

In the design of new computer systems there exists an enormous number of alternative decisions. In this thesis the major design parameters that will be allowed to vary are the number of processors(Pc's)† and memories(Mp's) and their relative speeds. The quantitative approach to performance evaluation consists of three major phases [GrenU72]:

(i) Collection of data: this phase involves the planning and conducting of the experiment for data collection as well as techniques for measurement.

(ii) Analysis of data: this phase consists of construction of models and estimation of parameters in the models as well as validation of the models.

(iii) Interpretation of data: this phase concerns the summarization of the results and new insights gained in the study as well as making decisions based on the results.

The emphasis of the application of quantitative methods is on the convergence of two aspects. The first aspect is the reliance on data either from experimentation on the real system or from simulation. The second is the use of

---

†We use the PMS notation of Bell and Newell [BellC71a] in this thesis to describe hardware organization.

mathematical models. It is easy to collect massive amounts of confusing data unless one has some model. On the other hand, a model without empirical validation is at best an intellectual exercise.

Since no performance measurements of the actual system can be made until it has been designed, implemented, and then observed over a long period of time, it becomes necessary to use analytic and simulation models. Analytic models enable the designer to explore a large design space quickly and rather economically. However, modeling is not an easy task and it is often necessary to simplify the model to make it amenable to mathematical analysis, remembering that any mathematical model is only an approximation of real-life events. If the system is too complex to allow a complete analytic study, the system behavior can be modeled at various levels of abstraction in a hierarchical fashion.

Simulation offers an different approach: probabilistic emulation of a mathematical model that portrays the aggregate bahavior of the real system. We gain in realism since we are no longer forced to impose assumptions for analytical convenience. However, simulations tend to be expensive if a high degree of realism on a detailed level is required. Due to the stochastic nature of simulation results, their precision can be measured by the standard deviation or confidence intervals of the estimates obtained. The confidence interval gets tighter as the size of the experiment is increased. The standard deviation is proportional to the square root of the length of the run. Hence, simulation studies are most valuable when focused on a small set of design alternatives

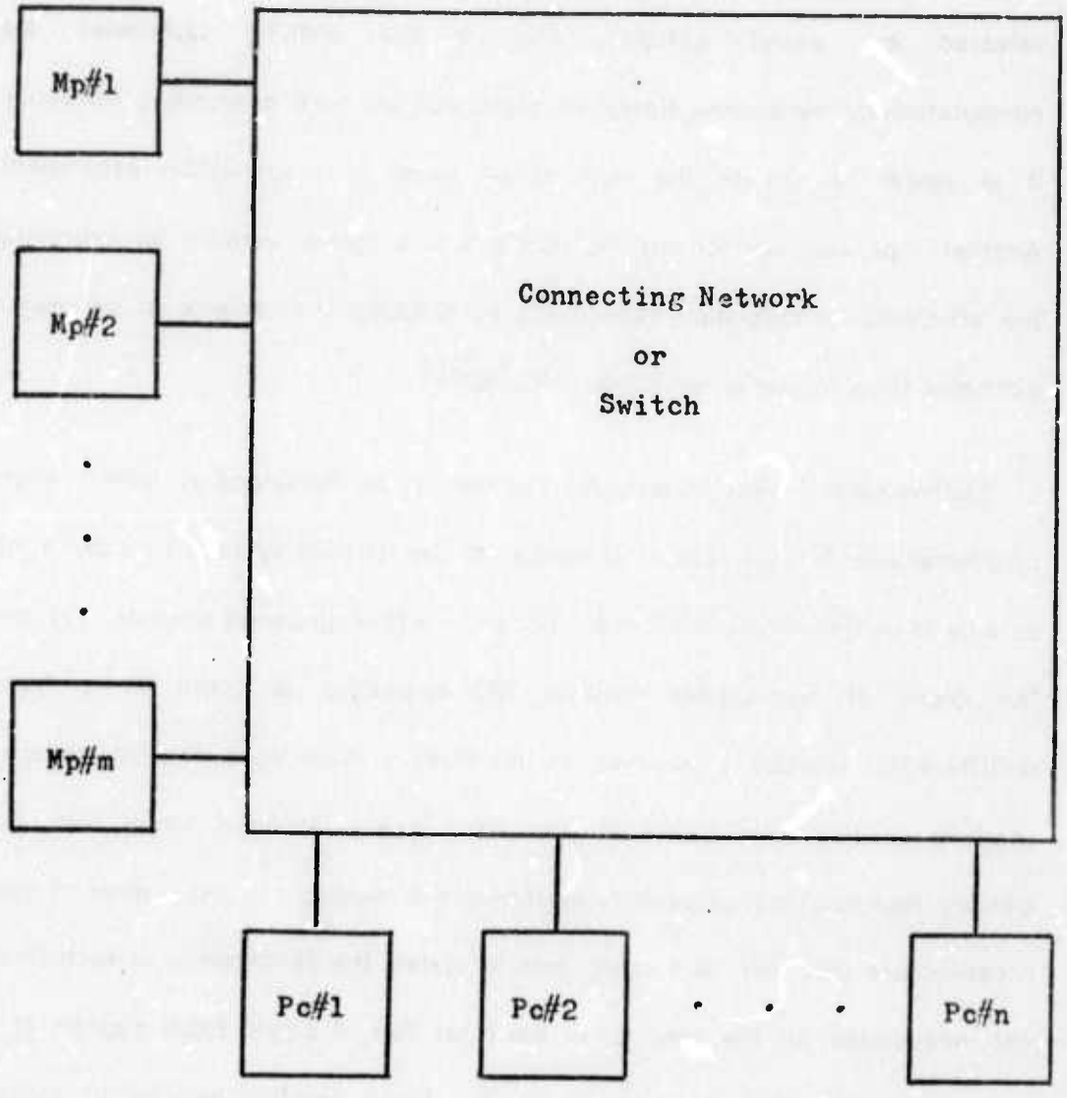
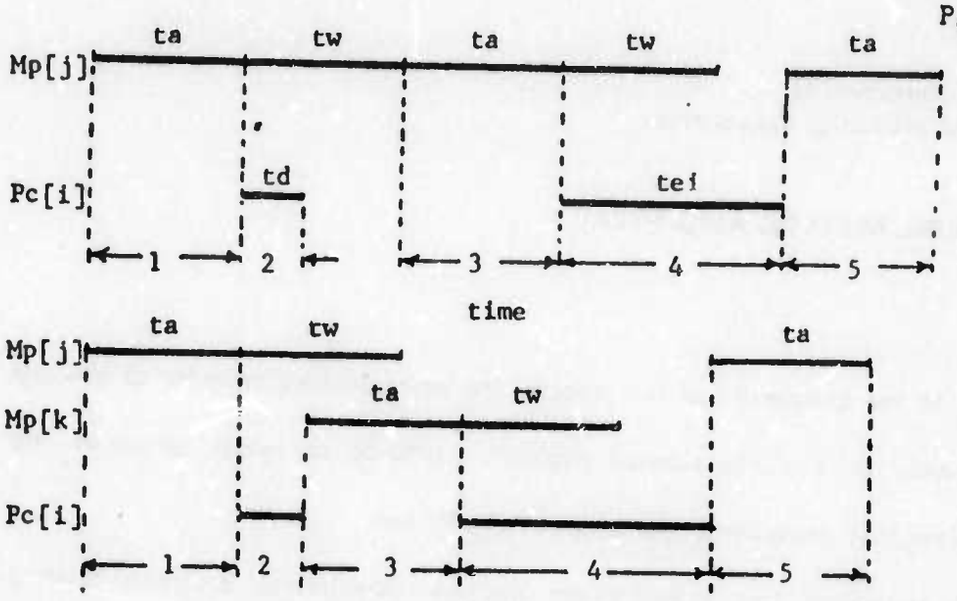


Figure 1.1 A simple block diagram of a multiprocessor system.

selected by analytic studies. Also, if the analytic techniques are computationally expensive, simulation might well be more economical. Moreover, it is easier to change the mathematical model in a simulation experiment. Another important use for analytic models is as a control variable for improving the efficiency of simulation experiments by reducing the variance of parameter estimates from simulation experiments [GaveD71].

Mathematical models of computer systems can be developed at various levels of abstraction. A large number of models for time-sharing systems consider a job as a basic unit[MckiJ69], and in many models of multiprogrammed computer systems the block of instructions between I/O operations is taken as a basic unit[BuzeJ71; GaveD67]. However, in this study a much more detailed model is used to analyze interference as processors access individual words from the memory modules. Each processor's performance is measured by the number of memory accesses per unit time. In a multiprocessor system the performance of each Pc is not independent of the behavior of the other Pc's. A simple block diagram of a multiprocessor system is shown in Fig. 1.1. The connecting network or switch provides a path form each of the n Pc's to each of the m Mp's, such that a connection between Pc[i] and Mp[j] does not hamper a connection between Pc[k] and Mp[l], where $i \neq k$ and $j \neq l$. The processors contend with each other for memory service. This contention is referred to as memory interference. This thesis presents a set of techniques for determining the extent of memory interference as measured by the average number of busy memories or the rate at which the Mp is accessed.



Legend:

1	instruction fetch	ta	memory access time
2	instruction decoding	tw	memory restore time
3	operand fetch	td	instruction decode time
4	instruction execution	tei	processor execution time
5	next instruction fetch		

Figure 1.2a: An example of the timing of a typical instruction.

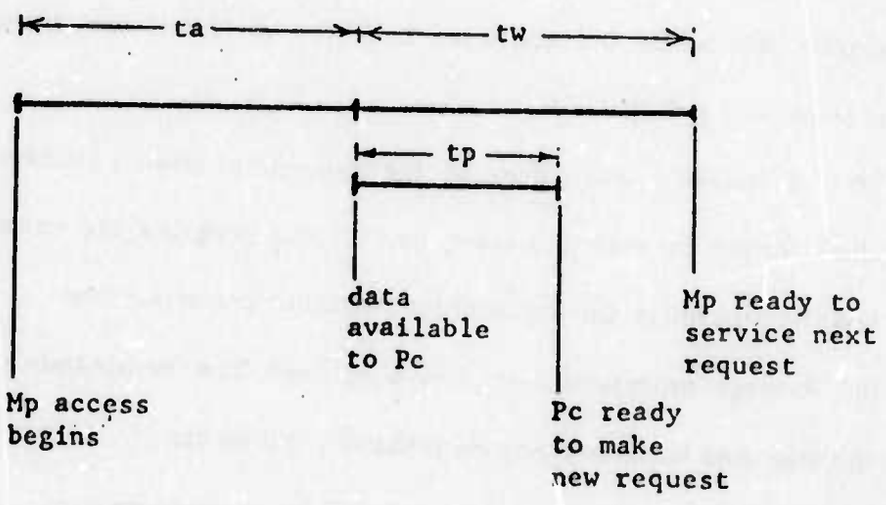


Figure 1.2b: Simplified processor behavior : unit instruction
Two units model the instruction shown in Fig. 2.1a.

1.1 GENERAL MODELING ASSUMPTIONS

Due to the complexity of the problem, the exact detailed behavior of memory interference in a multiprocessor system is difficult to model. Some of the parameters that characterize the behavior of a Pc are:

(i) Instruction mix : Instructions can be characterized by their relative frequency. In general, processor behavior varies for different instructions. However, in this thesis differences in instructions are not modeled explicitly. Processor behavior is modeled as an ordered sequence, consisting of a memory request followed by a certain amount of execution time. At this level of abstraction no distinction is made between the processing needed to decode an instruction and the processing corresponding to its execution. Thus, the processing time characterizing a Pc depicts only the aggregate behavior of the real Pc. Figure 1.2 depicts the actual and abstracted behaviors. A typical ***unit instruction***† is shown in Fig. 1.2b.

(ii) Probability distribution of the processing time: Instructions are characterized by their processing time. Typical programs are measured to find the probability distribution of the instruction processing time.

(iii) Average processing time: This is obtained from measurements similar to those used for determining the probability distribution.

---

†The concept of an unit instruction was first proposed by Strecker[StreW70].

(iv) Access pattern of a Pc: This is the trace of the pages or memory locations accessed by the Pc. In this study serial correlation between successive memory accesses will be ignored; not a very serious assumption since data and instruction references are intermingled. Demand patterns will be modeled as sequences of Bernoulli trials. Memory accesses will be characterized by the memory unit to which they are addressed. Let p_{ij} denote the probability that the i-th processor requests service from the j-th memory unit. Thus, the demand pattern of each processor is equivalent to a sequence of Bernoulli trials. Unless otherwise specified, p_{ij} will be assumed to be equal to 1/m, where m is the number of Mp's.

The effect of I/O activity will not be modeled explicitly. Strecker [StreW70] has shown that if the rate of I/O requests is RIO, then a fraction RIO/(m*tc) of the memory access rate can be apportioned to I/O.

A processor is said to be ***queued*** if it is waiting for or in the process of receiving memory service. A processor is said to be ***active*** if it is currently being serviced by a memory. Likewise, a memory is said to be ***occupied or busy*** if there is at least one processor queued for that memory unit.

Primary memory behavior is a function of the fabrication technology, i.e. core or semiconductor. Memory performance can be characterized by the ***access time*** (ta), ***rewrite time*** (tw), and ***cycle time*** (tc). Nominally, the cycle time is the sum of the other two. In this study, no distinction is made between read and

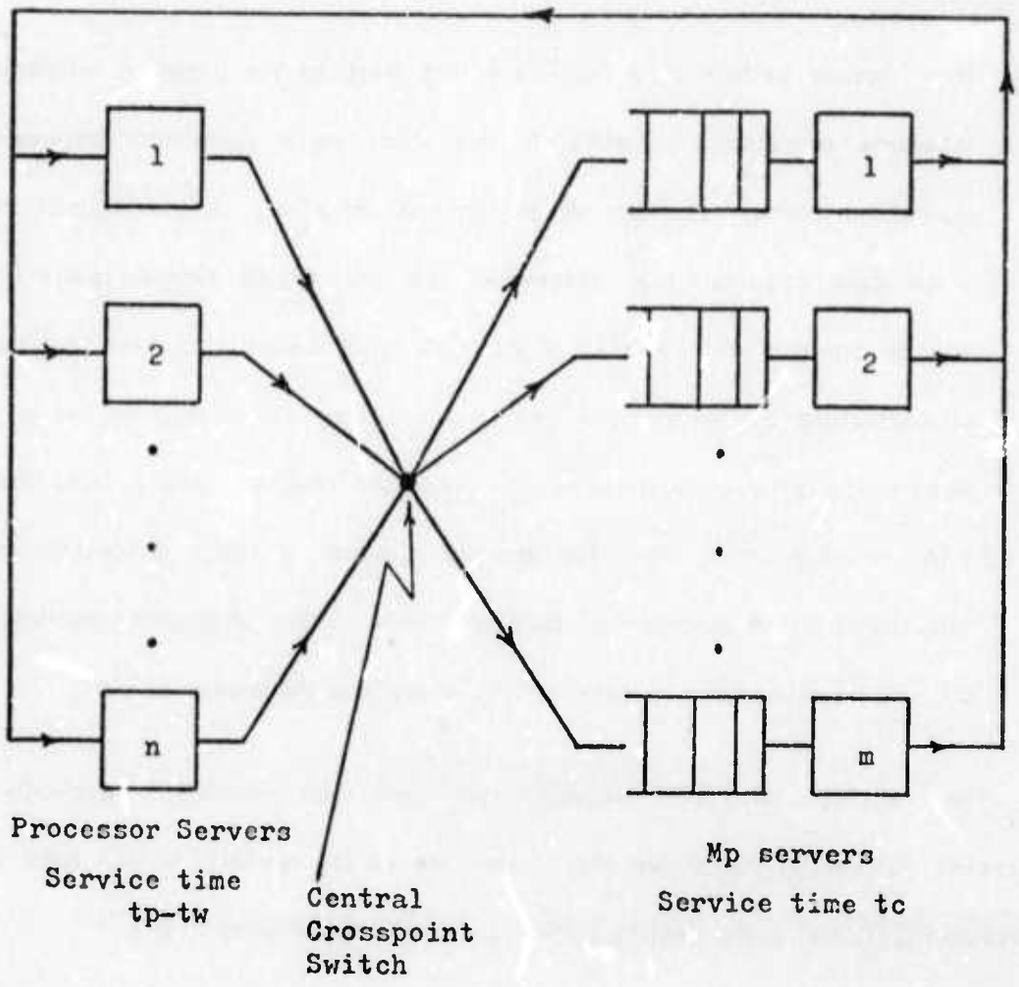


Figure 1.3 Structure of the queueing model

write operations. The effect of interleaving within a Mp module is to make the access and cycle times seen by a Pc variable. Most of the models in this thesis will use the average values.

The processing time shown in Fig. 1.2 is the effective processing time measured from the time when the Pc gets the data from its last memory access to the time when the next memory request reaches the memory. Thus, the delay associated with address mapping and communication protocol needed to make a request are attributed to the Pc. The memory access time includes the time required to set up the switch for the data transfer. If the memory control unit introduces some delay, then that is also added to the access time. A more detailed description is presented in Chapter 5.

1.2 MODELING CONCEPTS

A queueing model will be used to analyze memory interference. Figure 1.3 shows the basic structure of the model. All time delays are modeled as service centers and there is one job in the queueing system for every Pc. In real systems the Pc can start execution while the Mp is in its rewrite cycle. If $tp \geq tw$ the service time of the memory will be assumed to be tc, and the effective service time of the Pc will be assumed to be tp-tw. This simplification allows the Mp to start serving the next job as soon as the last job has left. The

overall system behavior is unaltered by this simplification. However, when tp<tw the queueing model cannot be used for reasons explained in detail in Chapter 4.

The ***number of Pc's*** will be denoted by n and the ***number of Mp's*** by m. The multiprocessor system will be referred to as a nxm system. The state of the system will be denoted by a vector describing the sizes of the various queues. The major technique used in this thesis involves Markov chains[ParzE62]. A brief review of some of the definitions and concepts is presented here.

A ***stochastic process*** is a family of random variables X(t), t∈T indexed by a parameter t varying in an index set T. The stochastic process is a ***discrete parameter*** process if T={0,1,2,...} or {0,±1,±2,...}. The process is a ***continuous parameter*** process if T={t≥0} or {-∞<t<∞}.

A discrete parameter stochastic process {X(t), t=0,1,2,...} or a continuous parameter stochastic process {X(t), t≥0} is said to be a ***Markov process*** if, for any set of n points $t_1<t_2<...<t_n$ in the index set of the process, the conditional distribution of $X(t_n)$, for given values of $X(t_1),...,X(t_{n-1})$, depends only on $X(t_{n-1})$, the most recent known value; more precisely, for any numbers $x_1,...,x_m$

$$P[X(t_n)\le x_n \mid X(t_1)=x_1,...,X(t_{n-1})=x_{n-1}] = P[X(t_n)\le x_n \mid X(t_{n-1})=x_{n-1}]$$

Intuitively, this means that, given the ***present*** of the process, the ***future*** is independent of its ***past***. The set of possible values of a stochastic process is called its ***state space***. The state space is called discrete if it contains a

finite or countably infinite number of states. A state space which is not discrete is called continuous. A Markov process whose state space is discrete is called a ***Markov chain.***

A Markov process is described by a ***transition probability function,*** denoted by $P(E,t \mid x,t_o)$, which represents the conditional probability that the state of the system will at time t belong to the set E, given that at time $t_o(<t)$ the system is in state x. The Markov process is said to have ***stationary transition probabilities,*** or to be ***homogeneous*** in time, if $P(E,t \mid x,t_o)$ depends on t and t_o only through the difference $(t-t_o)$.

A Markov chain is ***irreducible*** if every state can be reached from every other state not necessarily in one step. The ***period*** of a state i is defined as the greatest common divisor of all integers k such that the probability of returning to state i in k steps is greater than 0. A state of an irreducible Markov chain is ***aperiodic*** if it has period 1. A Markov chain is ***aperiodic*** if every state in its state space is aperiodic.

A discrete parameter irreducible aperiodic Markov chain that has stationary transition probabilities possesses a ***stationary state probability distribution.*** Let Z(k) denote the steady state probability of state k. Then,

$$Z(k) = \sum_j TRANS(k,j)*Z(j)$$

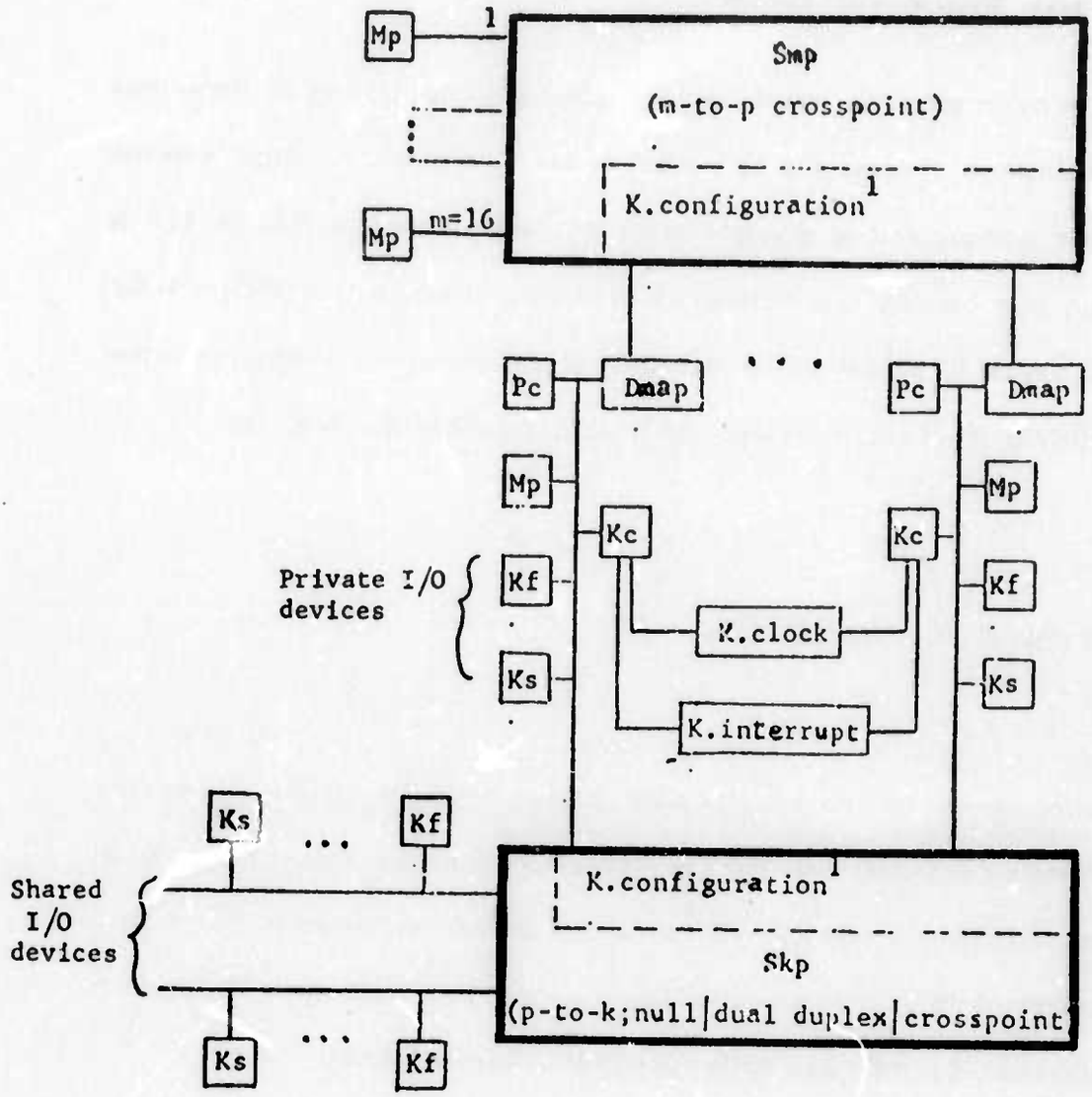
where TRANS(k,j) is the one step transition probability from state j to state k.

Transition probabilities from a ***current state*** to a ***next state*** will be

evaluated for the irreducible aperiodic discrete Markov chain models in the forthcoming chapters. The steady state probabilities will be used to calculate the average number of busy Mp's which is equal to the number of unit instructions executed in one memory cycle. The ***unit instruction execution rate (UER) or memory access rate (MpAR)*** is obtained by dividing the average number of busy Mp's by the cycle time.

1.3 EXTANT MULTIPROCESSOR SYSTEMS

A group at Carnegie-Mellon University is currently in the process of constructing a multiprocessor computer system (C.mmp) that will have up to sixteen central processors (PDP-11/20's) sharing the same physical address space [BellC71b; WulfW72] and concern has been expressed about the performance of such a system with this many active processors. The models developed in this thesis will be used to predict the performance of C.mmp in Chapter 5. Figure 1.4 illustrates the major components of a multiprocessor such as C.mmp. In addition to the processors, there is a set of memory modules that are able to operate independently; little would be gained if all the processors had to wait for service from a single memory module. Thus, between the processors and the memory modules (Mp's) is an n by m crosspoint switch, which allows any Pc to access any Mp. There are a number of ways of implementing the switch; Fig. 1.5(a) depicts



where: Pc/central processor; Mp/primary memory; T/terminals;
Ks/slow device control (e.g., for Teletype);
Kf/fast device control (e.g., for disk);
Kc/control for clock, timer, interprocessor communication
Dmap/relocation registers for mapping Pc address into Mp address space.

[1] Both switches have static configuration control by manual and program control

Fig. 1.4 Proposed CMU multiminiprocessor computer/C.mmp.

an n by m crosspoint switch, and Fig. 1.5(b) illustrates the use of trunk lines; combinations of these two basic schemes can yield many other other schemes. Other multiprocessors, although limited to a small number of Pc's, i.e. two to four, also basically use a crosspoint switch, e.g. the Burroughs D825[AndeJ62] and Univac 1110. For further discussion of trunk lines, and a variety of other switching structures, the reader is referred to Bell and Newell [BellC71a].

1.4 COMMENTS ON EARLIER WORK

A review of current literature shows very few models of memory interference. Skinner and Asher [SkinC69] proposed a discrete Markov chain model for multiprocessor systems with tp=tw. The analysis was presented for a small number of Pc's(≤2). However, for larger systems the complexity of the problem deterred the authors from further pursuit of an analytic solution.

Strecker[StreW70] developed a set of simple approximate models. Most of his modeling assumptions are similar to those used in this thesis. While the analysis of Skinner and Asher is rigorous and exact, Strecker's analysis is approximate. In this thesis, an exact analysis of a discrete Markov chain model for systems with tp=tw is presented. As expected, the exact analysis is very complex. However, the results of the exact analysis suggest more reasonable approximations that yield performance estimates that are more accurate than

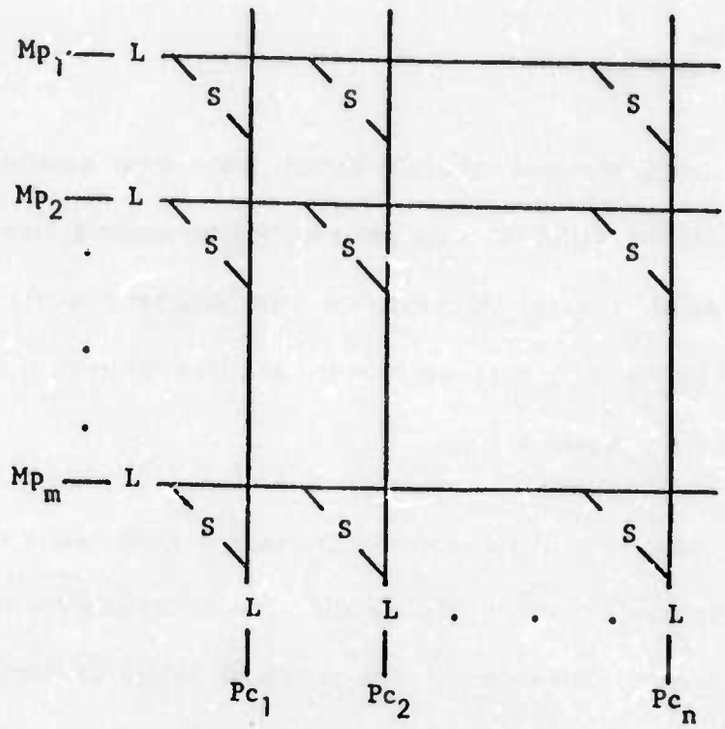


Figure 1.5a mXn crossbar switch

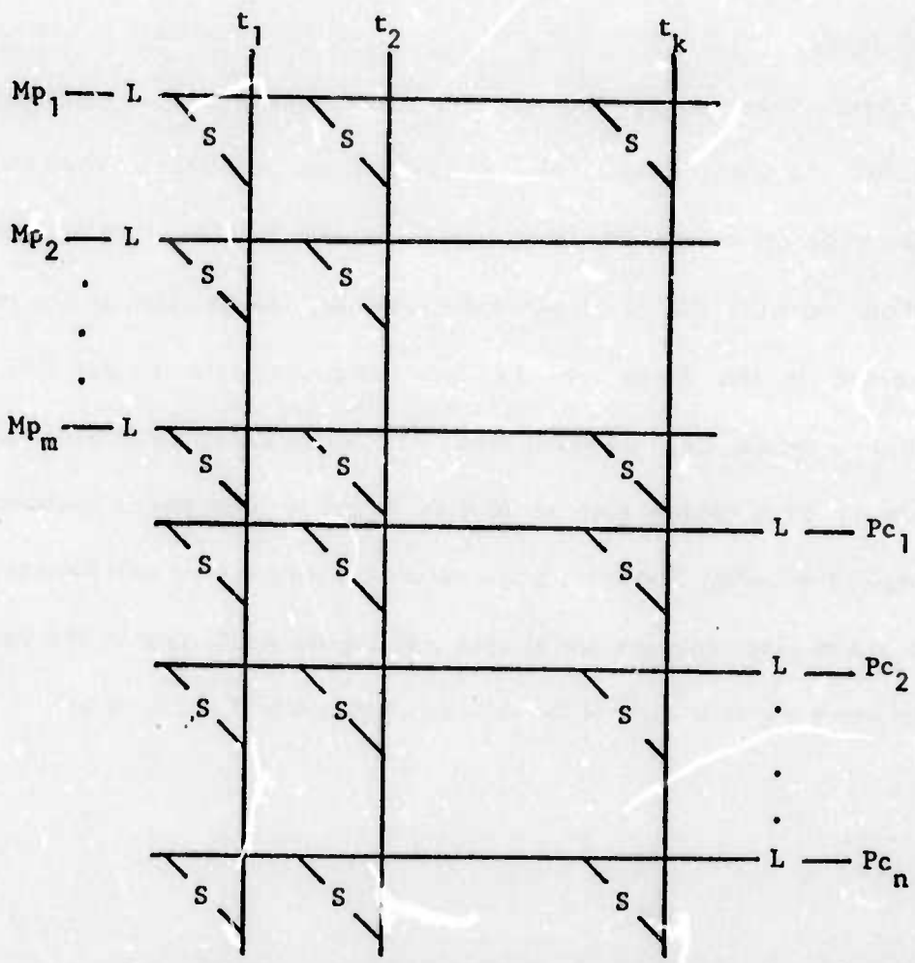


Figure 1.5b k-trunk line switch

Strecker's. For instance, the exact discrete Markov chain model described in Chapter 2 shows that the *MpAR* for a jxk and a kxj multiprocessor system with tp=tw is almost equal, a result not apparent from Strecker's work. Also, Strecker's formula for tp=tw is more accurate for m>n than for m<n; n is the number of Pc's and m the number of Mp's.

More detailed descriptions of the works of Skinner and Asher, and Strecker can be found in Chapters 2, 3 and 4. Bhatia[BhatS72] has shown how the results from memory interference models can be used as data for models of timeshared multiprocessor systems at the user program level.

A major contribution of this thesis is a systematic approach to the use of the Markov chain technique for analyzing memory interference in multiprocessor systems. The exact analysis of the Markov chain is complex. However, the behavior observed from the exact analysis is used to examine an approximate solution technique that is computationally simpler. Though some of the models presented in this thesis may be only marginally more accurate(5%) than Strecker's results, they may result in much more accurate estimates when used as inputs to other models such as Bhatia's model for time-shared systems. For example, the waiting time for a single server queueing system with Poisson input rate λ and exponential service at rate u is $1/(u-\lambda)$. A 5% error in the value of u can cause a greater error in the estimated waiting time if λ is close to u.

1.5 SYNOPSIS OF THESIS CONTENTS

The analytic models are described in detail in Chapters 2, 3 and 4. The models are mathematical abstractions of the real systems. Thus, they do not exactly reflect the true behavior of the physical system. However, the models will be referred to as exact or approximate depending on the quality of the technique used to analyze the mathematical model.

The models can be grouped into three broad classes : tp=tw, tp>tw and tp<tw, where tp denotes the average processing time. Systems witn tp=tw are described first because they represent boundary conditions for the other two cases. Different modeling techniques are examined for tp=tw and a reasonable approximation is proposed. A casual reader may find it useful to glance through the empirical results and validation of the models presented in Chapter 5 and the concluding remarks summarized in Chapter 6 before examining the mathematical intricacies of Chapters 2, 3 and 4. A more detailed summary of the thesis content is given below. Table 1.1 summarizes the salient characteristics of the various analytic models.

Chapter 2 is devoted to multiprocessor systems with tp=tw. A simple exponential server model provides some insight into the effect of adding a processor or a memory to the system. A more elaborate analysis for constant processing time uses discrete Markov chain techniques; an exact but unwieldy

TABLE 1.1

Salient Characteristics of Various Analytic Models

Model Descriptor	Processing Time	Memory Cycle Time	Remarks
MULTIPROCESSOR SYSTEMS WITH tp=tw			
Continuous Time Markov Chain	exponential	exponential	Jackson's Formulae are used to obtain a simple closed form solution.
Exact Discrete Markov Chain	constant	constant	The solution is algorithmic. Unwieldy for large systems.
Approx. Discrete Markov Chain	constant	constant	Approximation: non-active Pc's are reassigned to busy Mp's at the end of the cycle. Good for n≤m.
Strecker's Approximation	constant	constant	Simple closed form solution. Less accurate than above. Non-active Pc's are reassigned to all Mp's.
Skinner and Asher's Discrete Markov Chain	constant	constant	Exact discrete Markov chain analysis for upto 2 Pc's and m Mp's.
Approximate Model for Arbitrary P_{ij}	constant	constant	P_{ij} is not restricted to 1/m. Solution is simple but approximate.
MULTIPROCESSOR SYSTEMS WITH tp>tw			
Discrete Markov Chain for tp=tw+tc	constant	constant	Approximate analysis is presented. Queued Pc's are reassigned to all Mp's at the end of the cycle.
Discrete Markov Chain for Geometrically Distributed tp	geometric	constant	Approximate analysis. Prob[tp=tw+i*tc] = $\beta*\alpha^i$
McCredie's Exponential Server	exponential	exponential	Jackson's formulae are used. One Mp has different tc and different access probability. Cache.
Strecker's Analysis	constant	constant	Little's Formula is used.
Approximate Model for Systems with cache	constant	constant	Little's Formula is used. Cache memory speed is a parameter.
MULTIPROCESSOR SYSTEMS WITH tp<tw			
An Approximate Model	constant	constant	Model for tp=tw is used to obtain the conditional probability of Pc's second request going to an idle Mp, depending on the number of busy Mp's.
Strecker's Approximation	constant	constant	Results of tp=tw are usedto find the probabilty of request to an idle Mp. Average number of busy Mp's is used.

analysis and a simple approximate analysis is presented. This exact analysis of the Markov chain model is compared with Strecker's approximation. A new approximate model is introduced to analyze the effect of skewing the access patterns of the processors so that each has a greater preference for a different memory module.

Chapter 3 presents discrete Markov chain models for tp>tw. Techniques for an exact analysis of the models are introduced and some approximations suggested. Models are developed for constant processing time. McCredie's exponential server model[McCr73] and Strecker's approximate model for constant tp are discussed. Two new models for analyzing the effect of cache memories are also described.

Chapter 4 contains an approximate model for tp<tw and compares the results with Strecker's model.

Chapter 5 contains the results of some empirical measurements of PDP-11 programs. The processing time distribution is evaluated from these measurements. The process of extracting the abstract model parameters from the real physical system behavior is demonstrated. Predictions are made about the performance of Carnegie-Mellon University's C.mmp and compared with some actual measurements. This preliminary comparison with actual measurements shows the accuracy and utility of analytic models.

Chapter 6 summarizes the salient results developed in the thesis. An

example of the use of these models for examining design alternatives is also included. Some directions for future work are discussed.

CHAPTER 2

MULTIPROCESSOR SYSTEMS WITH TP=TW

In this chapter multiprocessor systems with tp=tw will be analysed. This could happen even with a very fast Pc. If the system is bus-bound and the Pc-Mp bus recovers at the same time that the memory is ready to service the next request, then the effective processing time (as seen by the memory) is equal to the memory rewrite time. With tp=tw, the analysis is simpler than with tp<tw and tp>tw. Also, it is a boundary condition for the other two cases. Thus, tp=tw is an interesting case for a preliminary comparison of various modeling techniques, even when tp is not equal to tw in reality.

A simple exponential server model provides some insight into the effect of adding a processor or a memory to the system. A more elaborate analysis for constant processing time uses discrete Markov chain techniques; an exact but unwieldy analysis and a simple approximate analysis is presented. This exact analysis of the Markov chain model is compared with Strecker's approximation. The results of the exact analysis are used to improve the accuracy of the approximate analysis. A new approximate model is introduced to analyze the effect of skewing the access patterns of the processors so that each has a greater preference for a different memory module. A diffusion approximation is also considered.

2.1 CONTINUOUS TIME MARKOV CHAIN MODEL

In our first model, we apply the classic simplifying assumption in queueing models: we model the service time, or cycle time, of the memory modules as exponentially distributed random variables [cf. WagnH69]. Clearly most memory systems do not have an exponentially distributed cycle time. However, techniques such as interleaving, cache memories, and the type of memory access(read, write, read-modify-write) suggest that this exponential assumption may be as good an approximation as the assumption that the memory cycle time is fixed, and not variable at all. Without further assumptions or approximations, we can use the results of Jackson [JackJ63], and Gordon and Newell [GordW67] to find the performance of the multiprocessor system. This technique is also used by McCredie [McCrJ73] for multiprocessors with tp>tw. This exponential server model is the simplest model to analyze. It also gives some basic insight into the extent of memory interference when the system has a large number of Pc's and Mp's.

Let the number of service centers be m. The states of the system are m-dimensional vectors with non-negative integer components, the j-th component representing the queue length at center j. If $K=(k_1,k_2,\ldots,k_m)$ is a state vector, then let $S(K)=\sum_{i=1}^{m} k_i$. Transition from one center to another is characterized by a routing probability r_{ij}, i.e. the probability of going to center j on completion of service at center i. Jackson [JackJ63] has obtained

the equilibrium joint probability distribution of queue lengths for a broad class of queueing-theoretical models representing a network of service centers. Customer arrivals are modeled as a generalized Poisson process [cf. WagnH69], whose mean arrival rate varies almost arbitrarily with the total number of customers already in the system. Service completions at each center are also modeled as generalized Poisson processes, the mean service rate (u) at each center varying arbitrarily with the queue length there. Note that in Jackson's model all customers are identical. Muntz and Baskett [MuntR72] have a more general queueing network model that allows different classes of customers to have different branching probabilities. Gordor and Newell [GordW67] have presented a solution technique for closed queueing systems, i.e. networks of queues in which the number of customers is constant.

For closed queueing systems, Jackson's formulae for obtaining the equilibrium state probabilities are listed below.

$$P(K) = w(K)/T(S(K))$$

where,

$$w(K) = \prod_{j=1}^{m} \prod_{i=1}^{k_j} [e(j)/u] \quad \text{for } j \in [1,m]$$

$$\text{where } e(j) = \sum_{i=1}^{m} e(i) r_{ij} \quad j \in [1,m]$$

$$T(K) = \sum w(K) \quad \text{summed over } K \text{ with } S(K)=n$$

But, with Pc requests distributed uniformly and with the bus-bound situation or tp=tw, the exponential server model reduces to m servers with

customers circulating with uniform routing probabilities i.e. $r_{ij}=p_{ij}=1/m$. Thus, e(j) denotes the average frequency of visits to service center j. Using the above formulae we get,

$$w(K) = (1/u)^n$$

$$T(K) = \binom{n+m-1}{m-1}(1/u)^n$$

$$P(K) = \left[\binom{n+m-1}{m-1}\right]^{-1} \quad \text{for all } K \text{ such that } \sum_{i=1}^{m} k_i = n$$

i.e. all the states of the system have equal probability. Physically, this indicates that states with greater congestion in the queues are as likely as evenly distributed queues. Note that the above analysis holds even when successive memory requests are correlated as long as the average access frequency $P_{ij}=1/m$. The probability that a particular Mp module is idle, Prob{Mp[i] is idle}, is the fraction of the total number of states that has $k_i=0$.

In other words,

$$\text{Prob\{Mo[i] is idle\}} = \frac{\text{number of ways of assigning n Pc's to m-1 Mp's}}{\text{number of ways of assigning n Pc's to m Mp's}}$$

Therefore,

$$\text{Prob\{Mp[i] is busy\}} = n/(n+m-1)$$

$$\text{E[number of busy Mp's]} = m*\text{Prob\{Mp[i] is busy\}}$$

$$= m*n/(m+n-1)$$

The above expression has a number of interesting properties: the expression is symmetric in m and n; it has a basic hyperbolic form, asymptotic to n as m gets large; and, if we let m=n the above expression becomes

$$n/(2-1/n)$$

and

$$E[\text{number of busy Mp's}] \rightarrow n/2 \qquad \text{for } n>>1$$

The final observation has important implications. It states that as multiprocessor systems grow to include more and more Pc's, we are not faced with a law of diminishing returns: no matter how many Pc's are used, if we have the same number of memory modules, we can expect half the processors to be active.

2.2 A SIMPLE DISCRETE MARKOV CHAIN MODEL

For this analysis let us assume that all the Pc's are characterized by a single constant processing time tp. In this model, the memory access and cycle time are constant. The exponential server model discussed above allows the memory cycle time to have a large range of values. However, though the cycle time is not a constant (as seen by a Pc) it certainly does not have an exponential distribution. The constant service time model is an attempt to de-emphasize the small variance in the value of the cycle time. Although the processing time is not a constant in reality, this approach yields fairly good

estimates of the *MpAR* as substantiated in section 2.7. Also, all the memory units are assumed to have the same cycle time tc and access time ta. Thus, the memory rewrite time is given by tw=tc-ta. If tp= tw then all memory units can be considered to be operating synchronously. Thus, during any memory cycle the number of active Pc's is equal to the number of busy Mp's.

In this section, a simple Markov Chain Analysis is presented for the case in which the processors request every memory with equal likelihood. A multiprocessor system with n Pc's and m Mp's is likened to an occupancy problem with n balls and m urns. Balls are randomly assigned to the m urns at the beginning of a memory cycle. At the end of the cycle one ball is removed from each urn. Thus if there are k non-empty urns during cycle s then k balls are available for assignment during the (s+1)-th cycle.

The state of the above mentioned process is defined by a m-tuple $(k_1,k_2,\ldots,k_m)$, where $\sum_{i=1}^{m} k_i=n$ and $0\le k_i\le n$ for all i. The number of distinct states of the system is given by the combination, $\binom{n+m-1}{m-1}$ i.e. the number of ways in which n identical balls can be assigned to m bins [FellW66]. However, since all the processors behave identically, a number of the distinct states are equivalent i.e. they have the same occupancy and have the same components. e.g. states (2,1,1), (1,2,1), (1,1,2) are equally likely. Thus, the reduced states are given by the different ways in which the number n can be partitioned into m parts. i.e. the unordered integer solutions to the equation $\sum_{i=1}^{m} X_i=n$ for $0\le X_i\le n$ represent equivalence classes of equally likely states. The number of such

TABLE 2.1

SOME PROPERTIES OF THE DISCRETE MARKOV CHAIN MODEL

Number of Pc's Number of Mp's	Total Number of States	Reduced States	Execution time for program
2	3	2	< 1 sec.
4	35	5	< 1 sec.
8	6435	22	2 sec.
10	92378	42	8 sec.
12	1352078	77	1 min.
16	300540195	231	1 hour

partitions (for n≤m) is asymptotic to

$$\frac{1}{4n\sqrt{3}} \exp[\pi(2n/3)\uparrow 0.5] \qquad \text{[cf. BeckE64]}$$

Also,

$$F(x) = \frac{1}{(1-x)(1-x^2)\ldots(1-x^k)}$$

$$= 1+\sum p(i)x^i$$

is an ordinary generating function of the sequence (p(0), p(1), ..., p(k)), where p(i) denotes the number of partitions of the integer i that have no part exceeding k, k≤i [LiuC68]. Table 2.1 shows the total number of states and the number of reduced representative states as a function of n.

Let the *representative state* Si denote the set of compositions of the number n that yield the same partition e.g. the compositions (2,1,1), (1,2,1) and (1,1,2) correspond to the partition of the number 4 which has two 1's and one 2. Further, let Si.j be the individual compositions of the partition typified by representative state Si and Si.1 be that composition which has its components arranged in monotone non-decreasing order, i.e. (2,1,1) for the above example. The algorithm shown in Fig 2.1 generates all the partitions of n with the components in monotone non-increasing order.

Let $X_{i,j}$ denote the probability of a transition from Sj to Si. Then, due to the symmetry of the problem,

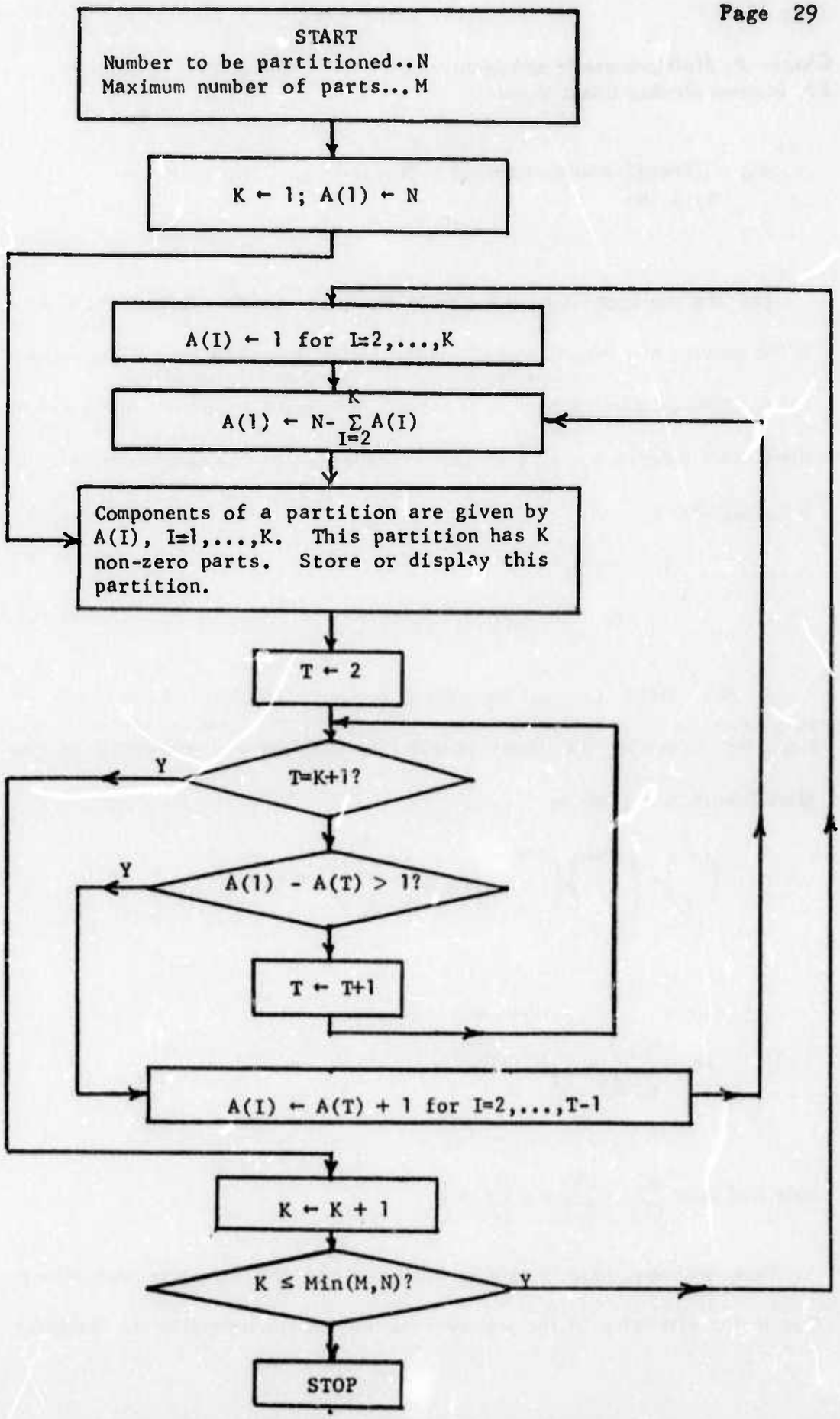


Figure 2.1 An algorithm for generating partitions

Chapter 2 : Multiprocessors with tp=tw
2.2 Discrete Markov Chain Model

$$X_{i,j} = \sum_{Si.k \in Si} \text{Prob}\{\text{Transition from } Sj.1 \text{ to } Si.k\}$$

Let the m-tuple $(k_1,k_2,\ldots,k_m)$ denote the state of the Markov chain. If x is the number of non-zero elements in this vector then at the end of the memory cycle, x new processors have to be reassigned to memory modules. At the end of the current memory cycle the queue is characterized by the *partial state* m-tuple $(j_1,j_2,\ldots,j_m)$, where

$j_i = k_i - 1$ if $k_i > 0$

$= 0$ otherwise.

A new state $(l_1,l_2,\ldots,l_m)$ is reachable from $(k_1,k_2,\ldots,k_m)$ if and only if $l_i \geq j_i$ for $1 \leq i \leq m$. If the above condition is satisfied the probability of the state transition is given by

$$\binom{x}{d_1} * \binom{x-d_1}{d_2} * \binom{x-d_1-d_2}{d_3} * \cdots * \binom{x-\sum_{i=1}^{m-1} d_i}{d_m} * (1/m)^x$$

where $d_i = l_i - j_i$

i.e. $$\frac{x!}{d_1!\, d_2! \ldots d_m!} * (1/m)\uparrow x$$

Note that since $\sum_{i=1}^{m} k_i = \sum_{i=1}^{m} l_i = n$, $\sum_{i=1}^{m} d_i = x$

Thus, we now have a formula for generating the transition probabilities. Due to the symmetry of the problem it suffices to generate only the transition

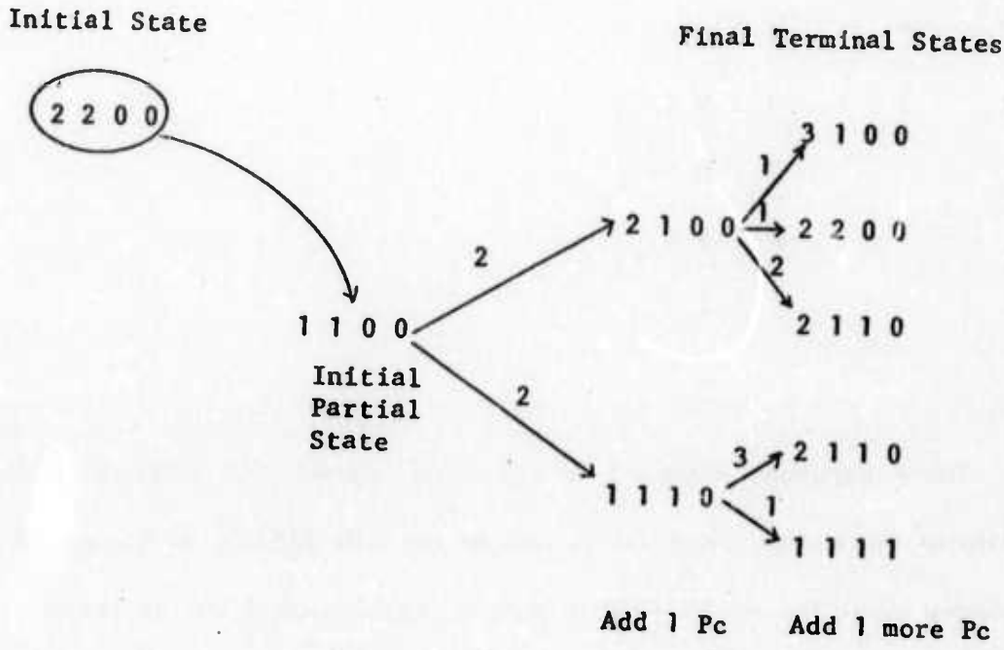


Figure 2•2 Next states accessible from initial state (2,2,0,0)

probabilities for the representative class of states. All the different ways of obtaining the same partition are lumped together to form a reduced state.

To illustrate a computational method** for generating the transition probabilities consider an example of a 4 by 4 system. The number 4 can be partitioned in 5 different ways as listed below:

4 0 0 0

3 1 0 0

2 2 0 0

2 1 1 0

1 1 1 1

These partitions represent 5 equivalence classes that characterize the state of the Markov Chain. Let us consider the state (2,2,0,0). At the end of a memory cycle, the resultant partial state is (1,1,0,0) with 2 free processors to be reassigned. Figure 2.2 shows the different ways in which these 2 Pc's can be assigned, one at a time, to reach a new partial representative state. After both Pc's are assigned a terminal state is reached. The number on the arrow indicates the number of ways of reaching the partial or terminal state that the arrow points to. Now the number of ways in which a final state can be reached from the

---

**The use of a tree to generate the transition probabilities was suggested by F. Baskett and D.Chewning of Stanford University.

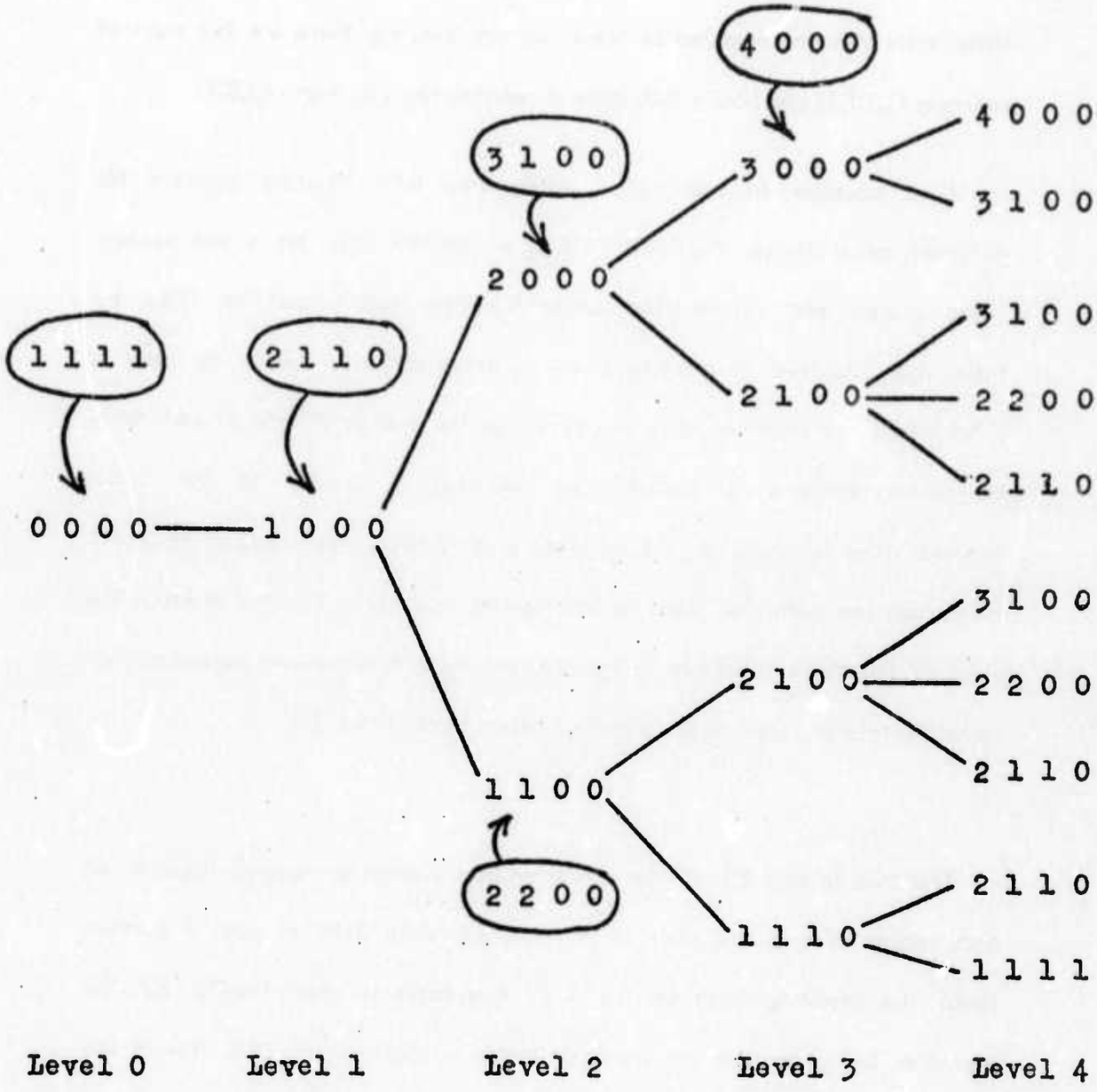


Figure 2.3 Enumeration tree for a 4 by 4 multiprocessor system.

initial state can be computed by traversing the tree, e.g. there are 2x1 ways of reaching (1,1,1,1) and (2x2 + 2x3) ways of reaching (2,1,1,0) from (2,2,0,0).

It is possible to construct a single tree with different pointers for different initial states. Figure 2.3 shows a complete tree for a 4x4 system. Initial states are circled. The entire transition matrix can be filled by traversing this tree. A convenient way of traversing this tree is by using a stack which has depth equal to one more than the number of Pc's. At each level the stack contains a partial state and has a pointer to the initial representative state (if any) from which it is derived. The stack is initialized to contain the path that leads to the topmost final state. For this example the stack is initialized as shown in Fig. 2.4, and Fig. 2.5 shows an algorithm† for using the tree to generate the transition matrix, shown in fig. 2.6.

The tree in Fig. 2.3 can be converted into a mesh by lumping together all occurrences of a partial state in the tree. e.g. state 2100 at level 3 appears twice. the resulting mesh for the 4 by 4 example is shown in Fig. 2.7. the algorithm for generating the transition matrix is shown in Fig. 2.8. Though the implementation of this algorithm†† involves a matrix multiplication and requires

---

†A FORTRAN implementation of this algorithm is listed in Appendix A-1.

---

††See Appendix A-2 for a listing of a FORTRAN implementation.

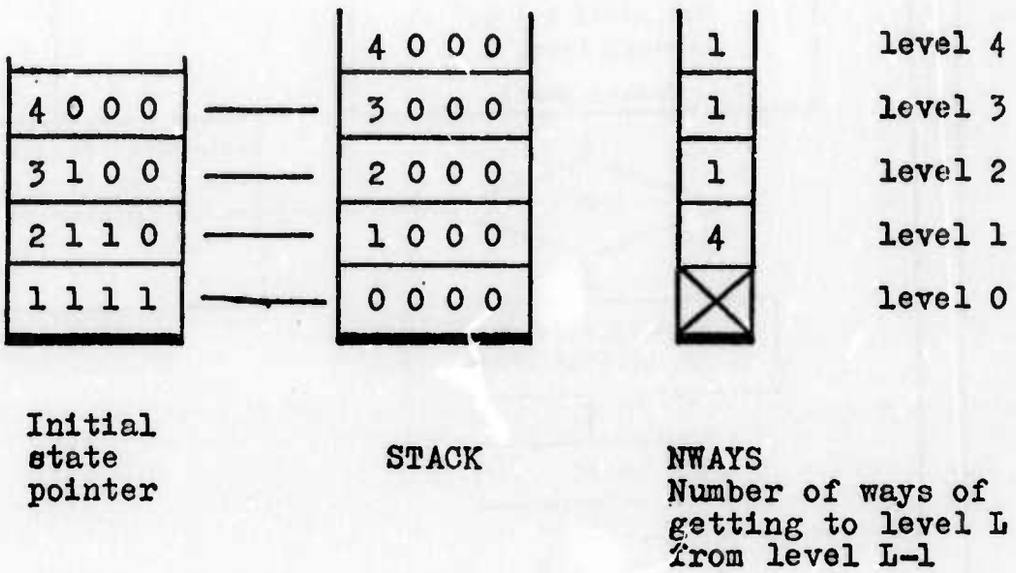


Figure 2.4 Initial contents of the stack for traversing the tree shown in figure 2.3

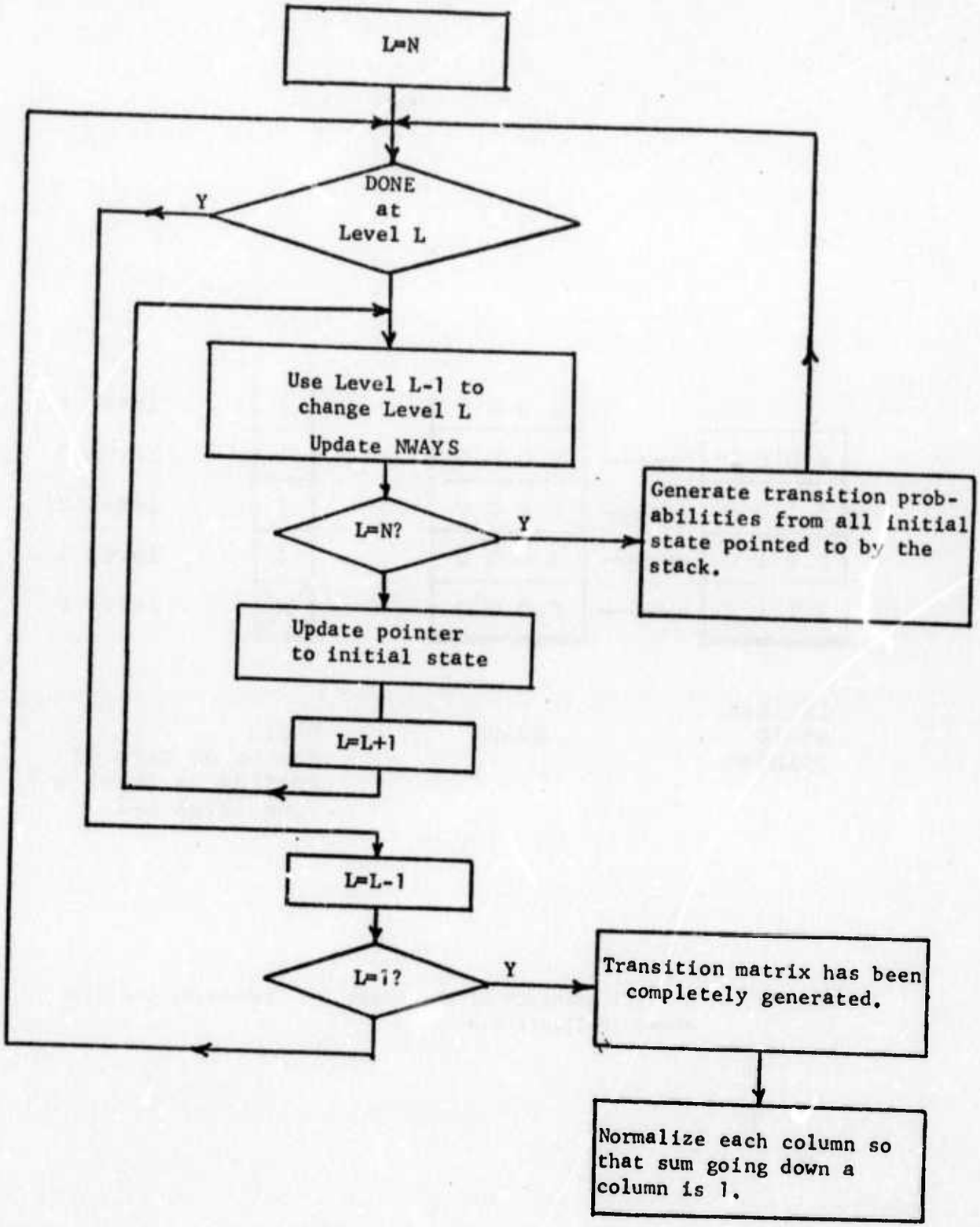


Figure 2.5 Algorithm for traversing the tree shown in Figure 2.3

	4 0 0 0	3 1 0 0	2 2 0 0	2 1 1 0	1 1 1 1
4 0 0 0	1	1	0	1	4
3 1 0 0	3	3+3	2	3+3+6	12+12+24
2 2 0 0	0	3	2	3+6	12+24
2 1 1 0	0	6	4+6	6+12+18	24+48+72
1 1 1 1	0	0	2	6	24

STEP 1 : Xij is the number of ways of reaching i from j. (obtained from the tree of fig.2.3)by using the

STEP 2 : $Xij = \frac{Xij}{\sum_i Xij}$ (Note that $\sum_i Xij = m^x$, where x of the m components of j are non-zero)

Final equations to be solved simultaneously :

$$\begin{bmatrix} P_{4000} \\ P_{3100} \\ P_{2200} \\ P_{2100} \\ P_{1111} \end{bmatrix} = \begin{bmatrix} 0.25 & 0.0625 & 0.000 & 0.015625 & 0.015265 \\ 0.75 & 0.3750 & 0.125 & 0.187500 & 0.187500 \\ 0.00 & 0.1875 & 0.125 & 0.140625 & 0.140625 \\ 0.00 & 0.3750 & 0.625 & 0.562500 & 0.562500 \\ 0.00 & 0.0000 & 0.125 & 0.093750 & 0.093750 \end{bmatrix} \begin{bmatrix} P_{4000} \\ P_{3100} \\ P_{2200} \\ P_{2100} \\ P_{1111} \end{bmatrix}$$

SUBJECT TO $P_{4000} + P_{3100} + P_{2200} + P_{2100} + P_{1111} = 1$

Figure 2.6 Steps in the generation of the transition matrix

more temporary storage it is faster than the algorithm in section 2 for large n. Thus, a space-time trade-off affects the selection of the algorithm to be used.

The following theorem and lemma can be used to increase the efficiency of the program that generates the transition probabilities.

Theorem 1: There is a one-to-one correspondence between a representative state and a partial state that the representative state reduces to at the end of a cycle.

Proof: Let $(k_1,k_2,\ldots,km)$ be a representative state. The partial state at the end of the cycle is given by

$$(j_1,j_2,\ldots,j_m)$$

where $j_l=k_l-1$ if $k_l>0$

$=0$ otherwise

Since no two representative states are alike and $\sum_{i=1}^{m} k_l=n$, it follows that the partial states are distinct.

Lemma : A partial state at level L in the enumerative tree of Fig. 2.3 can correspond to a terminal state with exactly n-L occupied Mp's.

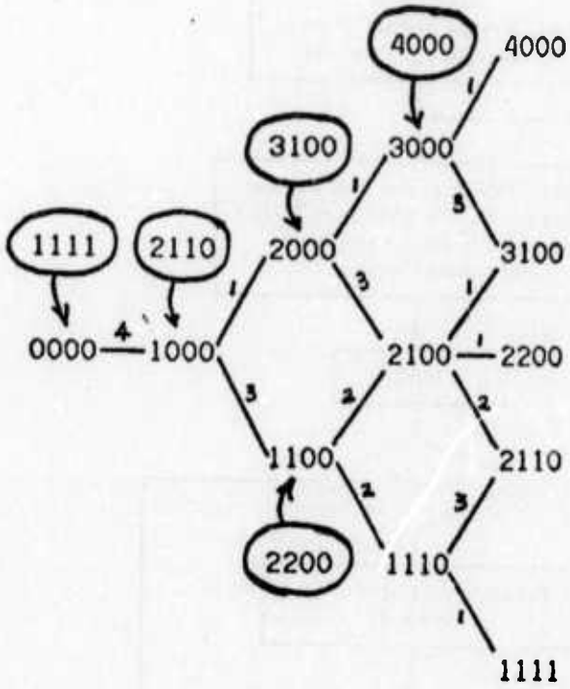


Fig. 2.7 Enumeration mesh for a 4 by 4 multiprocessor system.

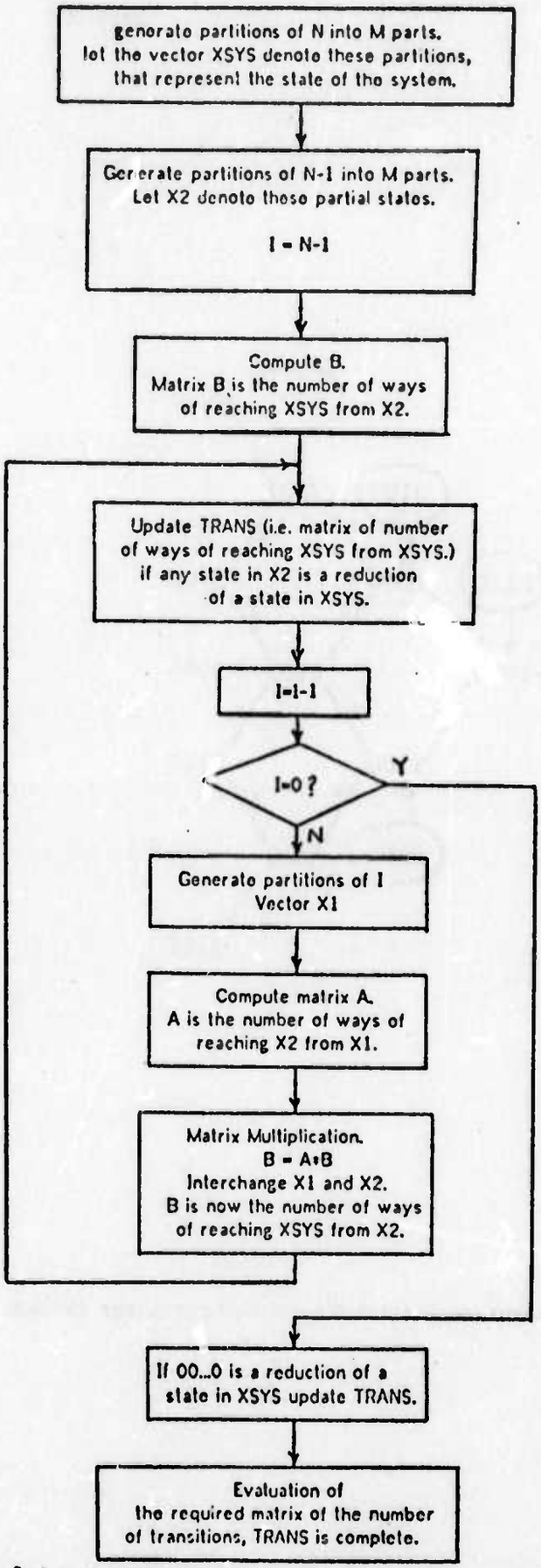


Fig. 2.8 An algorithm for evaluating the transition matrix.

Proof: Let $J=(j_1,j_2,...,j_m)$ be a partial state in the tree depicted in Fig. 2.3. Furthermore, let the number of non-zero elements elements in the partial state be y and let $\sum_{i=1}^{m} j_i=n-x$. Since one Pc is always removed from a non-empty queue at the end of a cycle, J is a partial state that can e reduced from a valid representative state $K=(k_1,k_2,...,k_m)$, if and only if

(i) The number of non-zero elements in K is x,and

(ii) x>y

Note that x and y are both less than or equal to min(m,n) and $\sum_{i=1}^{m} k_i=n$. Then, if x>y, J has at least x-y zeros. If x<y then there is no representative state K that corresponds to the partial state J. If x≥y, then the representative state is obtained by adding y 1's to the non-zero elements of J and replacing x-y zeros of J by 1. At level L, $\sum_{i=1}^{m} j_i = L$. Therefore, x, the number of occupied Mp's in K, is equal to n-L.

Figure 2.9 shows the average number of busy Mp's when n=m. The curve has an almost constant slope of .586 for n>4. Thus, this model also shows the absence of a law of diminishing returns. Figures 2.10 and 2.11 show the effect of adding a Pc and an Mp respectively on the average number of busy Mp's. Also, the average number of busy Mp's is almost symmetrical with respect to m and n. The results obtained from the model are compared with a less restrictive simulation model in section 2.7.

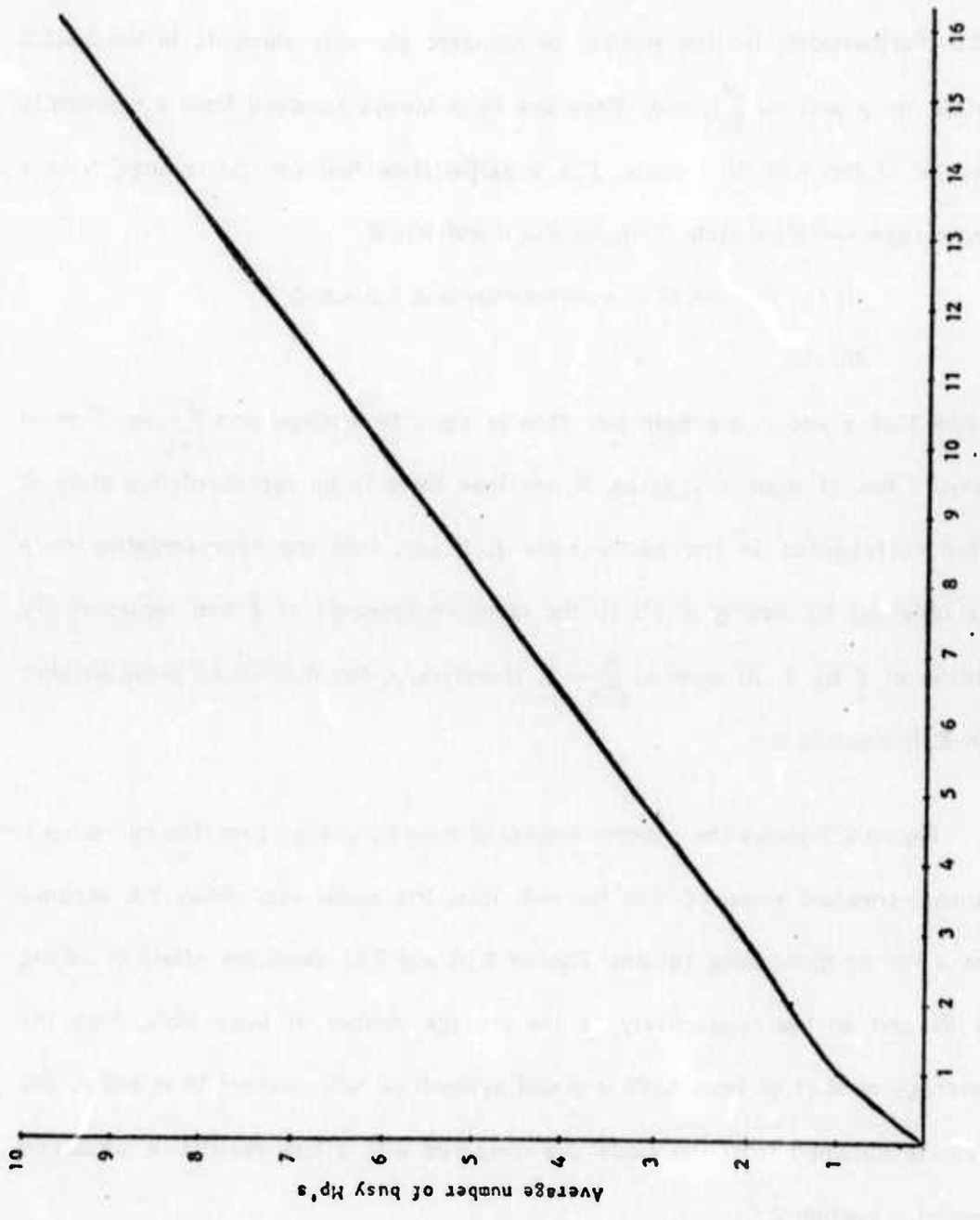


Figure 2.9 Multiprocessor Systems with n=m.

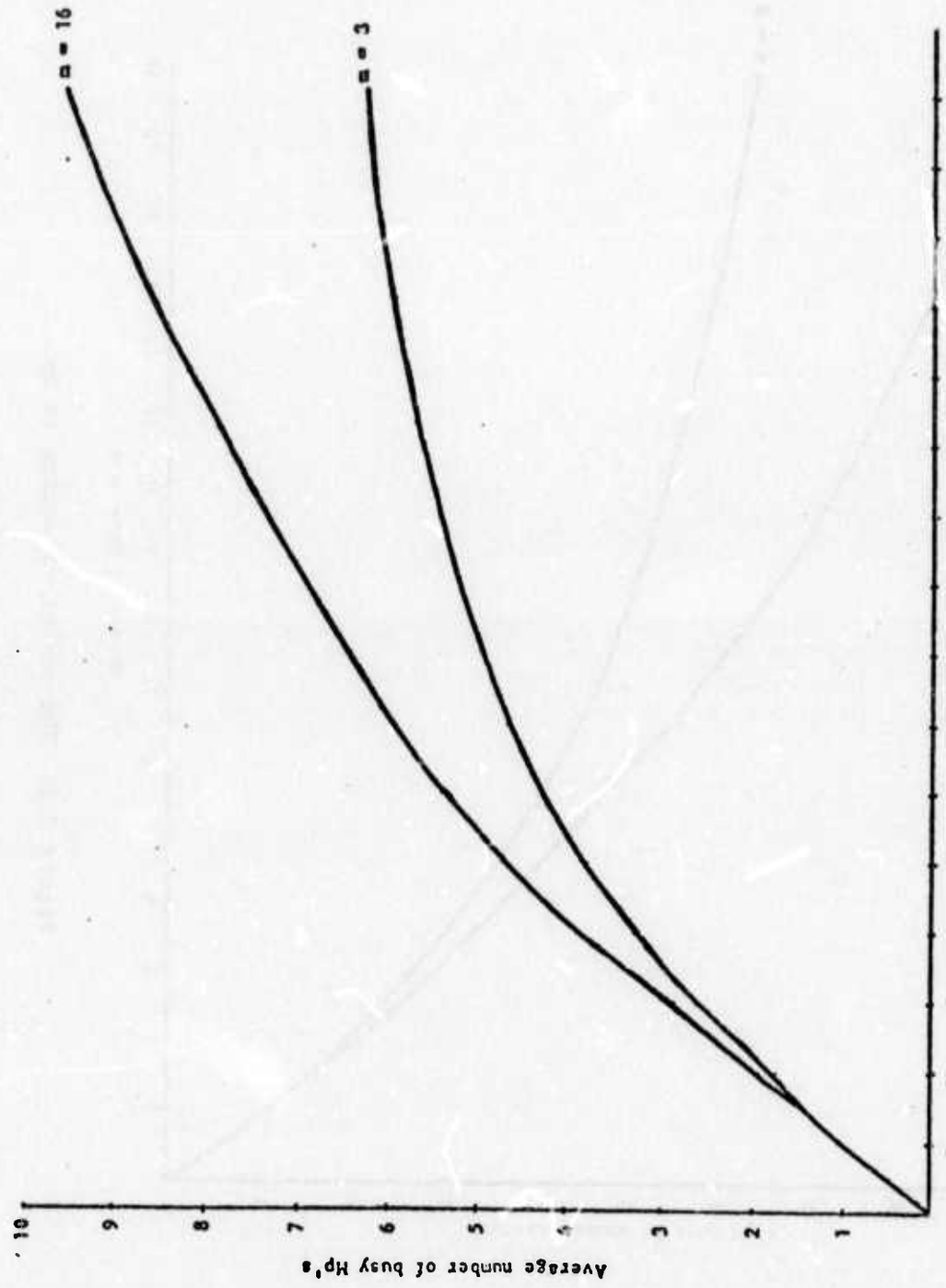


Figure 2.10 The effect of adding a Pc.

Figure 2.11 The effect of adding an Mp.

2.3 APPROXIMATE DISCRETE MARKOV CHAIN MODELS

Even with the representative state approach the number of states characterizing the Markov Chain increases rapidly as n increases. Table 2.1 shows the number of representative states as a function of n and the approximate execution time needed on a DEC PDP-10 for the FORTRAN program listed in Appendix A-2, which determines the stationary state probabilities. Though the analysis is exact, the size of the problem (as indicated by the array space used by the program) and the time required restricts the use of the model described in section 2.2.

2.3.1 A New Approximate Discrete Markov Chain Model

Because of the high cost of computation for the previous model, an approximate discrete Markov chain model will now be proposed, and the results of section 2.2 will be used to improve the applicability of this new approximate model. The state is denoted by the number of active Pc's. Thus the number of states is min(n,m). Note that only those Pc's that are active during the current cycle make new requests during the next cycle. Also, the number of busy memories is equal to the number of active processors. The approximation propounded here consists of removing the non-active Pc's from the Mp queues and reassigning them as indicated below. This approach was motivated by a gross intuitive feeling

that if the Pc's are removed from the queues and asked to make new requests, they would end up in the same queues as before. However, this is not exactly true, and the heavily congested states tend to be de-emphasized. Let the number of busy Mp's during the current cycle be i. Then, during the next cycle the i active Pc's make a new request to the m Mp's and some of the n-i non-active Pc's get serviced if they are at the front of the queue. However, in this approximate model, the n-i non-active Pc's are removed from the i Mp queues and reassigned to the same i queues. This is equivalent to the n-i Pc's making new requests to the i Mp's. Thus the i Pc's may not end up in the same queues that they were removed from. This approximation will be used widely in this thesis. The results of this section show the accuracy of the approximation.

Now, the probability that j out of the i active Pc's make a new request at the beginning of the next cycle to one of the i Mp's that are busy during the current cycle. is given by

$$XPROB = \binom{i}{j} * \left(\frac{1}{m}\right)^{j} * \left(1-i/m\right)^{i-j}$$

Thus, during the next cycle, with probability XPROB the n-i non-active Pc's and j active Pc's are assigned to the i busy Mp's of the current cycle, and the remaining i-j active Pc's make a request to the other m-i Mp's.

Let nn=n-i+j and k_1=min(nn,i). Note that nn denotes the number of Pc's that will be queued (during the next cycle) for the i Mp's that are busy during the current cycle. Also, let X(I_1) denote the conditional probability that I_1 out of i busy Mp's are also busy during the next cycle, given that n-i+j Pc's will be

queued for the i Mp's. The number of ways that nn different Pc's can be assigned to i different Mp queues is i^{nn} [RiorJ58, pp.90]. Also the number ways that the nn Pc's can access i Mp's so that exactly I_1 Mp's are occupied and $i-I_1$ are not is given by Riordan[RiorJ58] as

$$CM(i,I_1)*S(nn,I_1)$$

where $CM(i,I_1)= i(i-1)\ldots(i-I_1+1)$

and $S(nn,I_1)$ is the Stirling† number of the second kind

Thus,

$$X(I_1) = CM(i,I_1)*S(nn,I_1)/i^{nn}$$

Now, let $Y(I_2)$ be the conditional probability that I_2 out of the m-i currently non-busy Mp's are busy during the next cycle, given that i-j Pc's make a request. Then,

$$Y(I_2) = CM(m-i,I_2)*S(i-j,I_2)/(m-i)^{i-m}$$

Thus, the probability that $k=I_1+I_2$ Mp's will be busy during the next cycle is

$$XPROB*X(I_1)*Y(I_2)$$

Therefore, TRANS(k,i), the probability of a transition from current state i to next state k is

---

†Stirling Numbers of the second kind are used to convert from powers to binomial coefficients.

$$x^n = \sum_k S(n,k)\binom{x}{k} k!$$

Also,

$$S(i,j) = j*S(i-1,j)+S(i-1,j-1)$$
with $S(i,0)=S(0,i)=0$
and $S(i,i)=1$

TABLE 2.2

Comparison of Exact and Approximate Models

Approximate Discrete Markov Chain Model for tp=tw

Average Number of Busy Mp's

	m=2	m=4	m=8	m=16
n=2	1.5000	1.7500	1.8750	1.9735
n=4	1.8000	2.6550	3.2751	3.6291
n=8	1.9846	3.4858	5.0999	6.3680
n=16	1.9999	3.9343	6.8436	10.0058

Exact Discrete Markov Chain Model

Average Number of Busy Mp's

	m=2	m=4	m=8	m=16
n=2	1.5000	1.7500	1.8750	1.9735
n=4	1.7500	2.6210	3.2652	3.6268
n=8	1.8750	3.2657	4.9471	6.3149
n=16	1.9375	3.6270	6.3154	9.6258

$$\sum XPROB*X(I_1)*Y(i_2)$$

the summation is over the different ways of choosing I_1 and I_2 such that $k=I_1+I_2$.

A FORTRAN program that determines the steady state probabilities is listed in Appendix A-3. Table 2.2 shows the average number of busy Mp's as predicted by this approximate model. Due to the small number of states, this approximate model needs much less computer time; typically about 1 second of execution time on a PDP-10 for a 16x16 multiprocessor system. Table 2.2 shows that the average number of busy Mp's is almost symmetric in m and n. The approximate model has a larger error for n>m. Therefore, a better estimate of the performance of a nxm system can be obtained by evaluating the performance of a mxn system if n>m, a conclusion possible only due to the results of the exact analysis of section 2.2.

2.3.2 Strecker's Approximation

Strecker [StreW70] has an approximate closed form solution to the discrete Markov Chain model presented here. His approach is equivalent to removing the queued processors from all the memory modules at the end of a memory cycle and reassigning them among all the memory modules. In the approximate model proposed earlier in section 2.3.1, the Pc's that were queued at the end of the cycle were reassigned only among the busy Mp's. Thus, Strecker's analysis is more

TABLE 2.3

Expected number of busy memories in one cycle
Number of Pc's = 1,2,...,8 (rows)
Number of Mp's = 1,2,...,8 (columns)

Discrete Markov Chain Model

1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1.0000	1.5000	1.6667	1.7500	1.8000	1.8333	1.8571	1.8750
1.0000	1.6667	2.0476	2.2692	2.4095	2.5054	2.5748	2.6272
1.0000	1.7500	2.2701	2.6210	2.8630	3.0365	3.1657	3.2652
1.0000	1.8000	2.4102	2.8633	3.1996	3.4530	3.6482	3.8019
1.0000	1.8333	2.5059	3.0370	3.4533	3.7809	4.0415	4.2518
1.0000	1.8571	2.5751	3.1663	3.6486	4.0418	4.3636	4.6292
1.0000	1.8750	2.6274	3.2657	3.8024	4.2521	4.6294	4.9471

Strecker's Approximation

1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1.0000	1.5000	1.6667	1.7500	1.8000	1.8333	1.8571	1.8750
1.0000	1.7500	2.1111	2.3125	2.4400	2.5278	2.5918	2.6406
1.0000	1.8750	2.4074	2.7344	2.9520	3.1065	3.2216	3.3105
1.0000	1.9375	2.6049	3.0508	3.3616	3.5887	3.7613	3.8967
1.0000	1.9687	2.7366	3.2881	3.6893	3.9906	4.2240	4.4096
1.0000	1.9844	2.8244	3.4661	3.9514	4.3255	4.6206	4.8584
1.0000	1.9922	2.8829	3.5995	4.1611	4.6046	4.9605	5.2511

Percentage Error

0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	4.9979	3.1012	1.9082	1.2658	0.8941	0.6602	0.5100
0.0000	7.1429	6.0482	4.3266	3.1086	2.3053	1.7658	1.3874
0.0000	7.6389	8.0782	6.5484	5.0631	3.9299	3.1002	2.4935
0.0000	7.3856	9.2063	8.2680	6.8340	5.5463	4.5157	3.7114
0.0000	6.8548	9.6812	9.4685	8.2991	7.0191	5.8896	4.9512
0.0000	6.2507	9.7244	10.2214	9.4335	8.2900	7.1521	6.1450

approximate and will underestimate the interference. However, Strecker obtains his approximate solution in a closed form, which will be modified here to yield more accurate estimates of the *MpAR*. Thus the state of the system is considered independent of the state during the last cycle. If we use this assumption the distribution of Pc's queued for an Mp follows the binomial distribution:

$$\text{Prob}\{Y=r\} = \binom{n}{r}\left(\frac{1}{m}\right)^r\left(1-1/m\right)^{n-r}$$

where Y is a random variable equal to the number of Pc's queued for Mp[j] and $p_{ij}=1/m$ for all i and j.

Thus,

Prob{Mp[j] is busy} = 1- Prob{nobody is queued for Mp[j]}

$$= 1-(1-1/m)^n$$

In other words, the occupancy of Mp[j] is $1-(1-1/m)^n$, and

$$\text{E[no. of occupied Mp's]} = \sum_{j=1}^{m}\{\text{Occupancy of Mp[j]}\}$$
$$= m*[1-(1-1/m)^n]$$

Table 2.3 shows a comparison of Strecker's results and the exact Markov chain analysis. Note that Strecker's results are optimistic estimates of the unit execution rate. It is encouraging to note that such a simple expression is within 6 to 8% of the exact Markov Chain model for m/n>0.75. This is because his analysis assumes that all n Pc's always make a new request at the beginning of

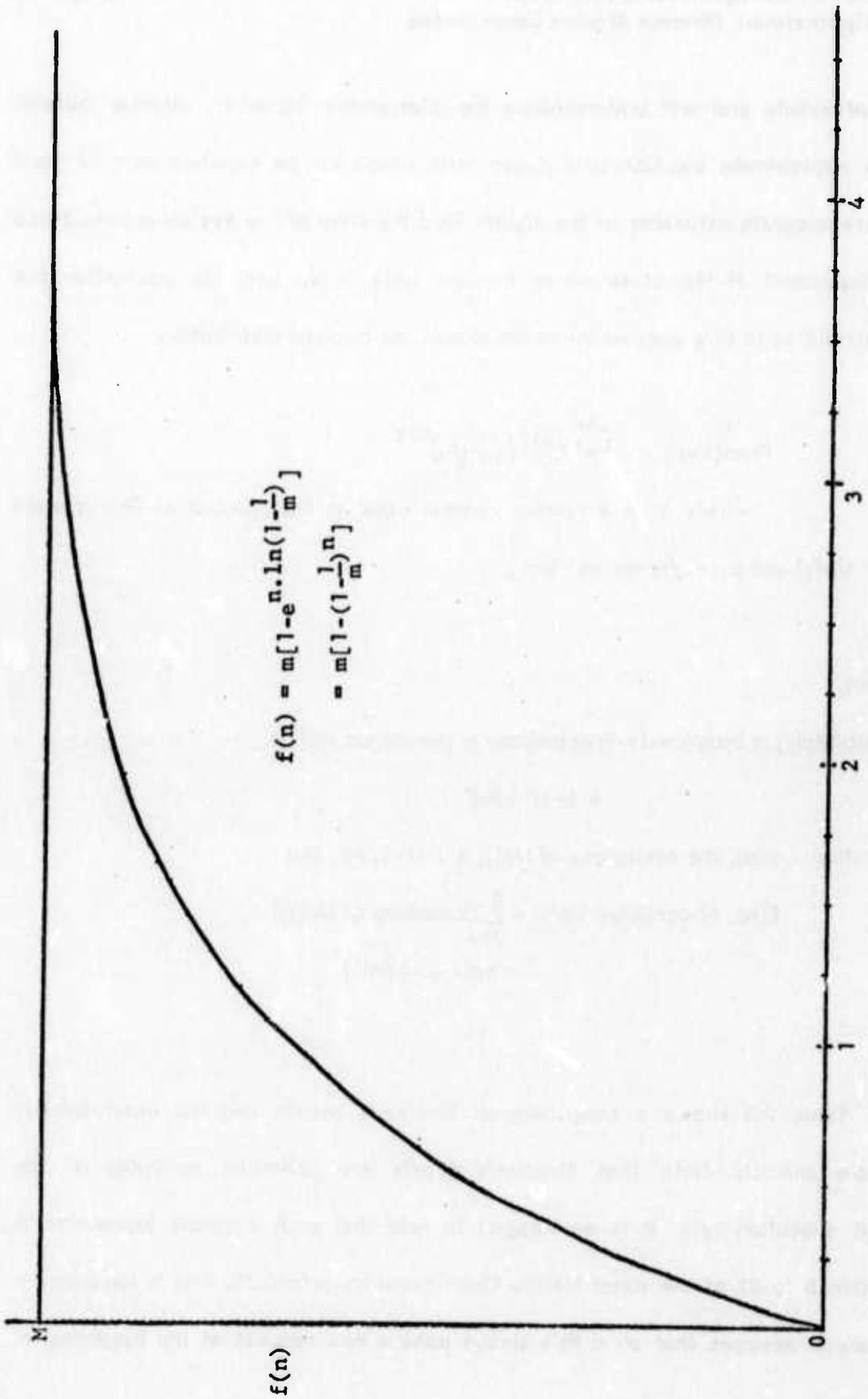


Figure 2.12 Strecker's formula for fixed m

each memory cycle, whereas in the discrete Markov chain only those Pc's that receive service are allowed to make new requests. Moreover, note that the expression $m*[1-(1-1/m)^n]$ can be written in an exponential form as

$$m*\{1-\exp[n* ln\ (1-1/m)]\}$$

Figure 2.12 shows a plot of the above expression for fixed m; the relaxation time $[ln\ (1-1/m)]^{-1}$ approaches m as m gets large.

The exact discrete Markov chain model of section 2.2 shows the performance to be almost symmetric in n and m. Also the analysis of the error of Strecker's approximation, shown in Table 2.4, indicates a greater accuracy for n<m. Thus, a more accurate estimate of the average number of busy Mp's is $i*[1-(1-1/i)^j]$, where i=max(n,m) and j=min(m,n). Note that the above formula was not derived by Strecker. It was possible to obtain it due to the knowledge gained from the exact analysis presented in section 2.2.

2.4 DISCRETE MARKOV CHAIN MODEL OF SKINNER AND ASHER

Skinner and Asher [SkinC69] model the multiprocessor system with tp=tw as a discrete Markov chain. They assume a matrix of probabilities that express the likelihood that a given processor requests service from a given memory at the beginning of a memory cycle, provided the Pc is not queued. They also assume a matrix of probabilities that express the likelihood of the various outcomes that

can arise when there are simultaneous requests to one memory by several processors. The state of the system is characterized by the processors queued for the different memory modules. A state transition matrix is formed from the access probabilities and the steady state probabilities of various states are determined by solving the state transition equations. The number of states of the system increases very steeply with an increase in the number of Pc's and Mp's. Closed form solutions are presented only for cases with up to 2 Pc's and n Mp's. The analysis in the previous section is similar to Skinner and Asher, but with uniformly random access patterns for all the Pc's, i.e. $p_{ij}=1/m$ for all i. The results of Skinner and Asher are compared with a new approximate model in section 2.6.

2.5 DIFFUSION APPROXIMATIONS

An approximation method that has been proposed for the solution of general queueing networks is the diffusion approximation [cf. NeweG71; KobaH73]. A discrete-state process is approximated by a Wiener-Levy diffusion process with a continuous path. The key assumption in such an analysis is that incremental changes in the queue lengths are normally distributed. This leads to a characterization of the queueing network by a set of diffusion equations. The accuracy of the approximation depends on three factors: (i) approximation of a

discrete-state process by a time-continuous Markov process, (ii) choice of proper reflecting barriers, and (iii) discretization of the continuous density function for queue lengths. Surprisingly, for the simple discrete Markov Chain model of section 4, the diffusion approximation yields a result identical to that with exponential servers derived from Jackson's formulae. However, the main utility of the diffusion approximation in this context is that it can be used to analyze the effect of different coefficients of variation (ratio of standard deviation to the mean) for the service time distribution. Unfortunately, for $P_{ij}=1/m$, the diffusion approximation predicts that the average number of busy memories to be independent of the service time distribution as long as all servers are identical. Thus, the diffusion approximation has proved to be a disappointing tool in this study.

2.6 AN APPROXIMATE MODEL FOR ARBITRARY P_{ij}

In this section, we explore the effect of non-uniform access probabilities (i.e. P_{ij} is no longer restricted to be equal to 1/m) on the ***MpAR***. This situation often arises in physical systems in which each Pc has a greater preference for a different memory module.Now, we analyze multiprocessor systems in which P_{ij} can take any arbitrary value between 0 and 1 subject to $\sum_{j=1}^{m} P_{ij}=1$. Let Qij denote the probability that processor i is queued for memory j. The probability that memory j is busy or occupied is given by

1-Prob(no processor is queued for memory j)

$$1-\prod_{i=1}^{n}(1-Qij)$$

assuming that the event of a Pc not being queued for an Mp is independent of other Pc's not being queued. The simulation results shown later justify this assumption.

The above denotes the average number of requests serviced in one memory cycle.

Thus, if, M is the expected value of the number of occupied memories during a memory cycle,

$$\text{then } M=\sum_{j=1}^{m}\text{Prob[Mp[j] is occupied]}$$

$$=\sum_{j=1}^{m}[1-\prod_{i=1}^{n}(1-Qij)]$$

In general, the probabilities Qij and Pij are not equal. The Pij's are a characteristic of each processor and therefore independent of the behavior of the other processors in the system. However, any Qij is a function of all the Pij's of the multiprocessor system. Strecker (StreW70) has evaluated the unit execution rate of a multiprocessor system in which Pij=1/m for all values of i and j. He makes no attempt to obtain a relationship between Qij and Pij. Strecker assumes that Pij=Qij and states that his results are approximate. The approximation in his analysis is due to the assumed binomial distribution for

the queued processors. If all the Pc's are identical and have equal likelihood of accessing every memory unit, then the probability of any processor being queued for any memory is uniformly equal. Since the memories operate synchronously and all requests occur at the end of a memory cycle a processor is always queued. Hence, the probability Qij=1/m.

Let us focus our attention on Pc[i] and Mp[j]. The time spent by Pc[i] in queue for Mp[i] depends on P_{ij} and Q_{lj}, $l \neq i$. Thus, Q_{ij} depends on other Q_{lk}'s, which in turn depend on Q_{ij}. Let us for a moment allow other processors to make requests to memory before Pc[i]; and let Y_{ij} denote the probability that none of the other n-1 Pc's request service from Mp[j].

Thus,

$$Y_{ij} = \prod_{l \neq i}^{n} (1-P_{lj})$$

Now, if none of the other Pc's make a request to Mp[j] the waiting time (including service) is one cycle time. However, if other Pc's make a request to Mp[j] before Pc[i], then Pc[i] has to wait for those Pc's to be served. Now, let us look at Pc[l], which has to wait for service from Mp[j] if other processors make a request before it does. Here, we shall allow other Pc's to make requests before Pc[l]. Thus, Pc[l] waits in queue for Mp[j] with probability $P_{lj}*[1-Y_{lj}]$. Thus, the average number of Pc's that Pc[i] finds waiting before itself is $\sum P_{lj}*[1-Y_{lj}]$. However, Pc[i] accesses Mp[j] with probability P_{ij}. Therefore, the weighted waiting time T_{ij} for Pc[i] in queue for Mp[j] is

$$P_{ij} * \{Y_{ij} + [1 + \sum_{l \neq i}^{n} P_{lj}*(1-Y_{lj})] * [1-Y_{ij}]\}$$

Therefore, the average time is equal to $\sum_{j=1}^{m} T_{ij}$; and

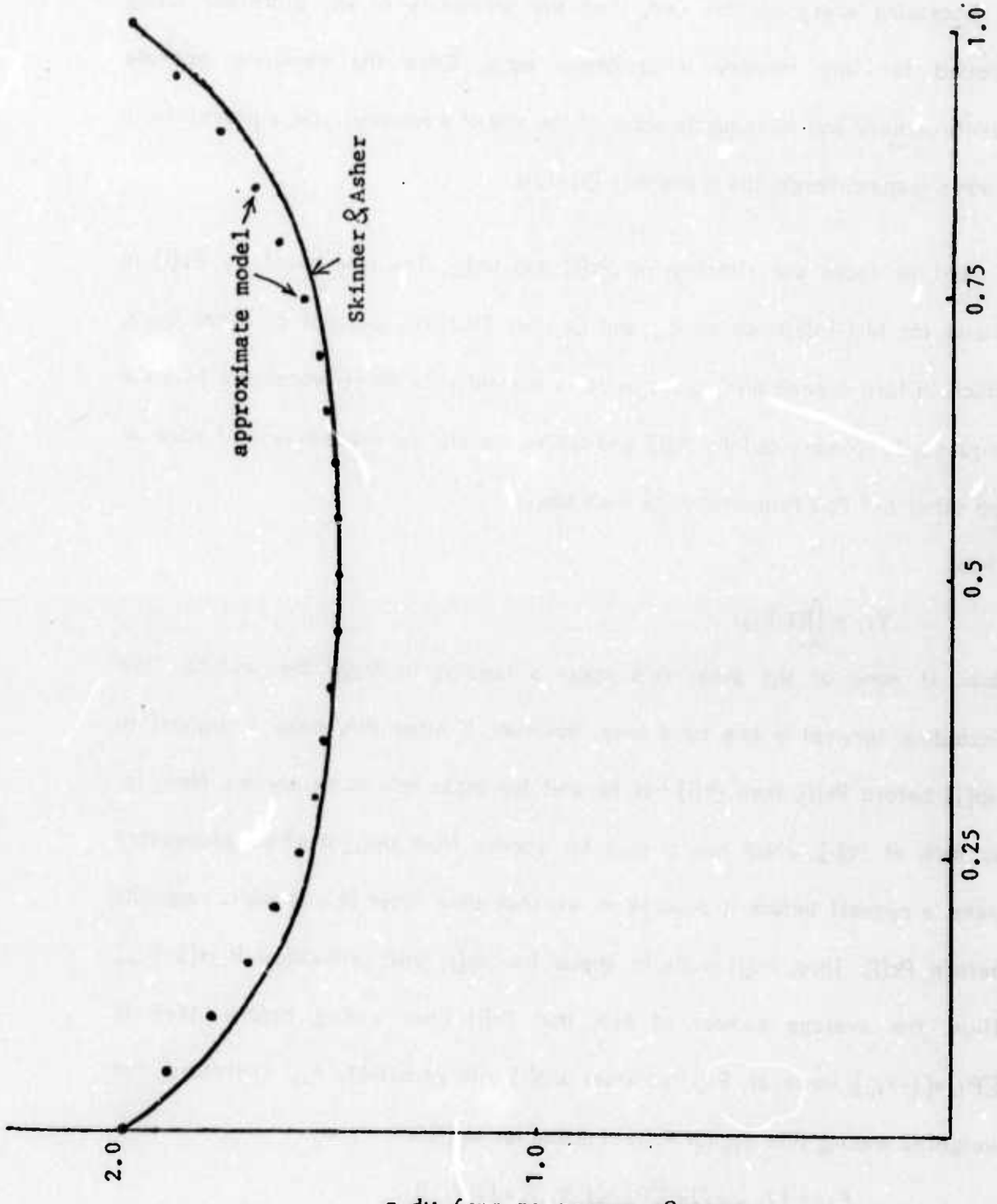


Figure 2.13 Comparison of approximate model and Skinner and Asher's results

$$Q_{ij} = T_{ij}/T$$

Figure 2.13 compares the execution rate predicted by the approximate model of this section with the exact Markov chain model of Skinner and Asher for a 2x2 multiprocessor system. P_{ij} is equal to α for i=j and equal to β for i≠j.

Now, suppose each Pc has a greater preference for one Mp. Let us use the model to examine the effect of assigning access probabilities so that $P_{ij}=\alpha>1/m$ for i=j and $P_{ij}=\beta=(1-\alpha)/(m-1)$ for i≠j. Note that $\beta<1/m$. Thus, each Pc has a greater preference for a certain memory module. For example, the access probability matrix for a 4x5 multiprocessor system is shown below.

$$\begin{matrix} \alpha & \beta & \beta & \beta & \beta \\ \beta & \alpha & \beta & \beta & \beta \\ \beta & \beta & \alpha & \beta & \beta \\ \beta & \beta & \beta & \alpha & \beta \end{matrix}$$

Figures 2.14 and 2.15 show the effect of changing α from 0 to 1 for a 8x16 and a 16x16 multiprocessor system. Also, Table 2.4 compares the results predicted by the model with simulation results, because the analysis is approximate. Note that both graphs show that the execution rate is a minimum for $\alpha=1/m$. With $\alpha=1/m$ this model predicts the same result as Strecker's approximation as both models assume a binomial distribution for the queued processors. The approximation error reduces as α increases, the error being zero for $\alpha=1$. An error correction factor can be used as described below. For $\alpha=1/m$

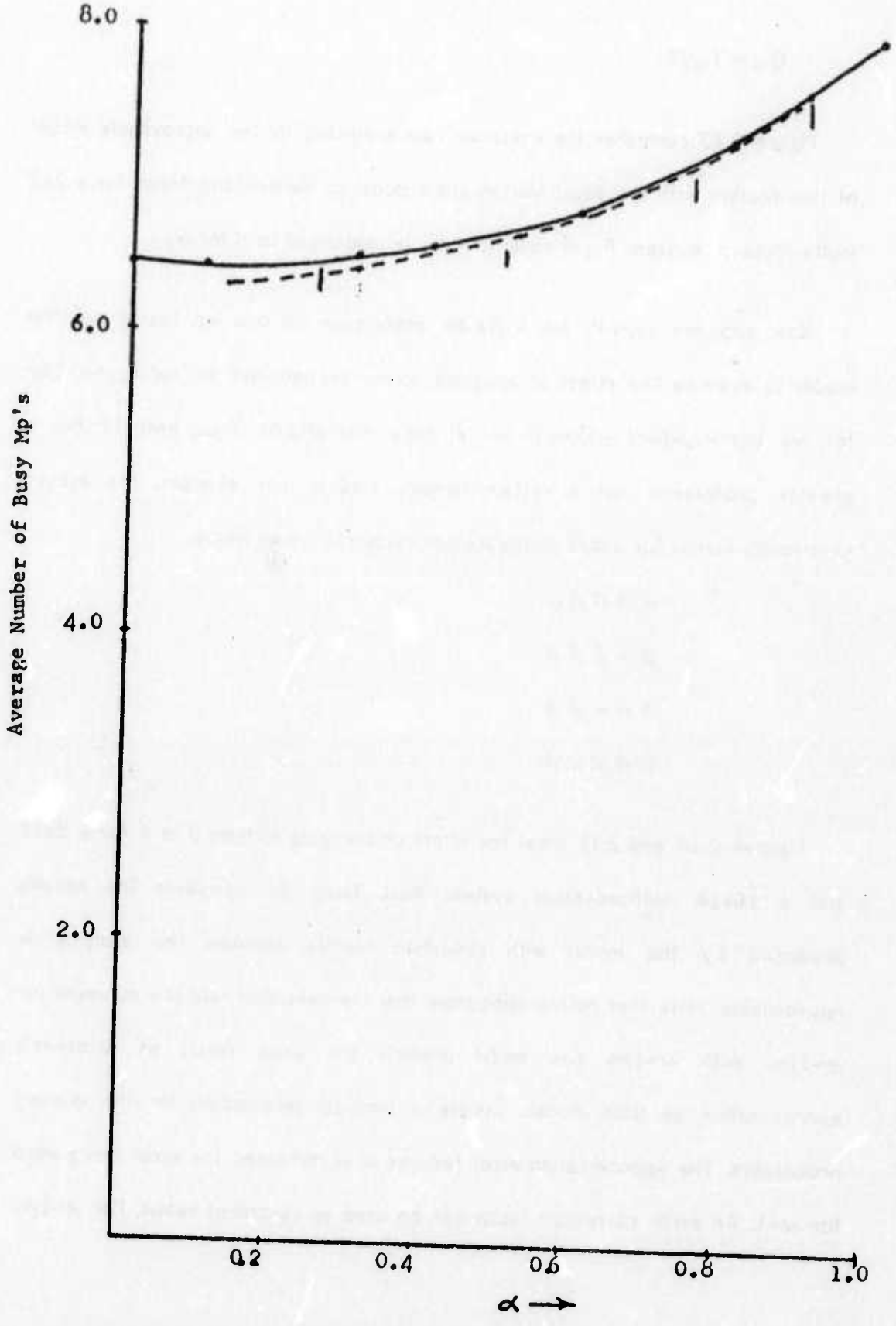


Figure 2.14 The Effect of α on the execution rate of a 8x16 Multiprocessor System.

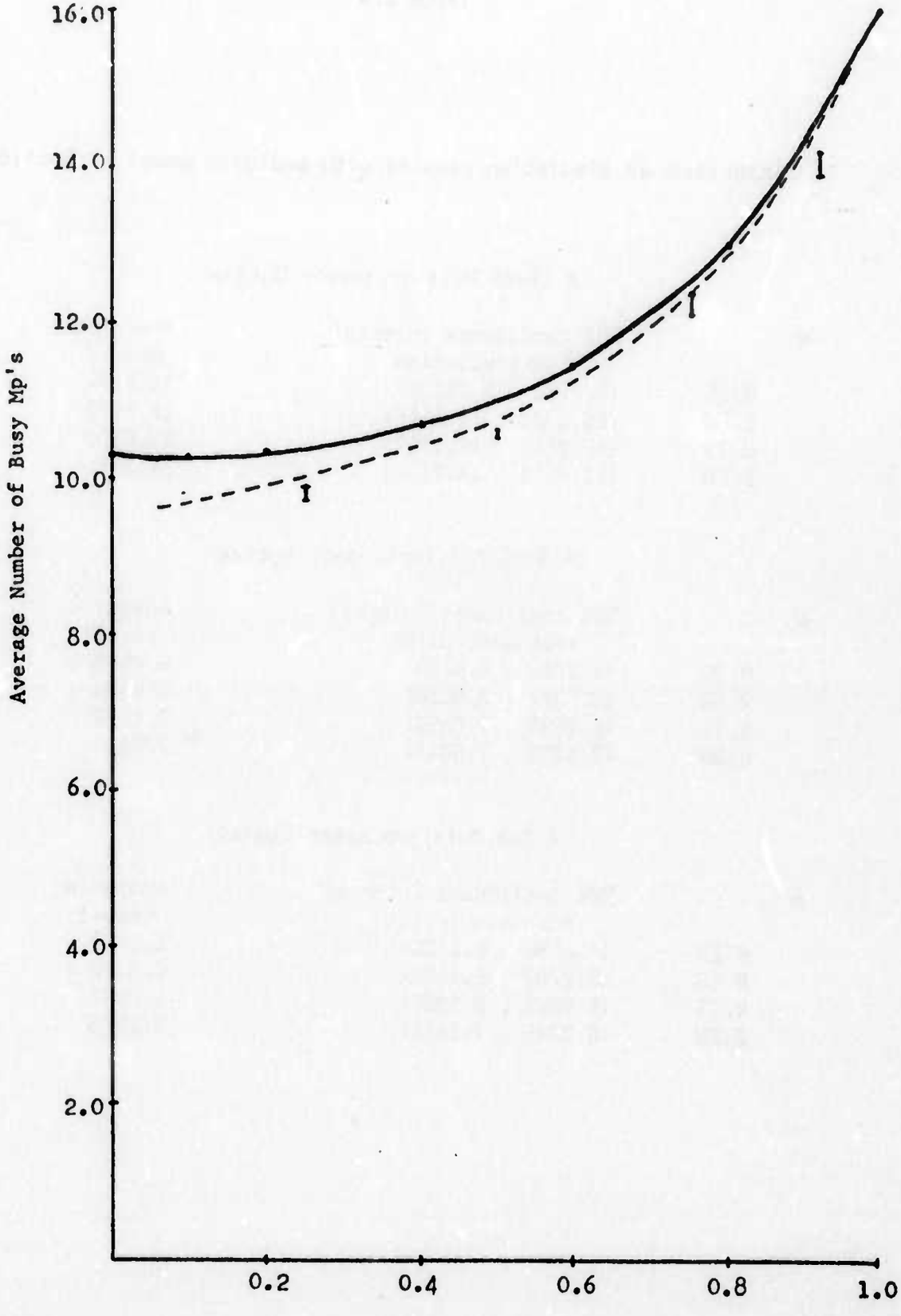


Figure 2.15 The Effect of on the execution rate of a 16x16 multiprocessor system.

TABLE 2.4

Comparison of simulation results with analytic model of Section 2.3

A 16x16 Multiprocessor System

α	90% confidence interval from simulation	Analytic Result
0.25	(9.7482 , 9.8329)	10.4174
0.50	(10.5824 , 10.5946)	10.9973
0.75	(12.0875 , 12.3907)	12.5323
0.90	(13.9111 , 14.2145)	14.3873

A 8x16 Multiprocessor System

α	90% confidence interval from simulation	Analytic Result
0.25	(6.2803 , 6.4139)	6.4880
0.50	(6.5374 , 6.5614)	6.6885
0.75	(6.9693 , 7.0853)	7.1673
0.90	(7.4338 , 7.6041)	7.6328

A 8x8 Multiprocessor System

α	90% confidence interval from simulation	Analytic Result
0.25	(4.8886 , 5.1126)	5.2798
0.50	(5.2797 , 5.4127)	5.5372
0.75	(6.0926 , 6.1901)	6.2807
0.90	(6.9245 , 7.1411)	7.1975

the exact Markov chain model should be used to compute the execution rate X_1; the approximate model of this section predicts an execution rate, X_2, equal to $m*[1-(1-1/m)^n]$ for $\alpha=1/m$. The correction factor for $\alpha=1/m$ is X_2/X_1. Let $e=(X_2-X_1)/X_2$. Then a linear error correction factor F is $1-e*(1-\alpha)/(1-1/m)$ for $\alpha>1/m$. The corrected estimate is F*X, where X is the execution rate for the given value of α. The dotted lines in figures 2.14 and 2.15 show the corrected execution rates. The vertical lines show 90% confidence intervals obtained by simulation†. This model shows the increase in the *MpAR* due to deskewing of the processors' access patterns.

2.7 CONCLUDING REMARKS

Tables 2.3 and 2.5 compare the numerical results obtained from the different models described. Note that the continuous and discrete Markov chain models exhibit similar trends, though the numerical values differ. Strecker's approximation gets better as m/n increases, whereas the continuous time and discrete Markov models get closer for larger n/m ratios. Table 2.6 shows some simulation results obtained with exponential distributions for the processing

---

†All confidence intervals in this thesis will be shown by vertical lines. Unless specified otherwise, the confidence intervals are calculated from about 10 independent samples, each averaged over about 3000 cycles.

TABLE 2.5

Expected number of busy memories in one cycle
Number of Pc's = 1,2,...,8 (rows)
Number of Mp's = 1,2,...,8 (columns)

Discrete Markov Chain Model

1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1.0000	1.5000	1.6667	1.7500	1.8000	1.8333	1.8571	1.8750
1.0000	1.6667	2.0476	2.2692	2.4095	2.5054	2.5748	2.6272
1.0000	1.7500	2.2701	2.6210	2.8630	3.0365	3.1657	3.2652
1.0000	1.8000	2.4102	2.8633	3.1996	3.4530	3.6482	3.8019
1.0000	1.8333	2.5059	3.0370	3.4533	3.7809	4.0415	4.2518
1.0000	1.8571	2.5751	3.1663	3.6486	4.0418	4.3636	4.6292
1.0000	1.8750	2.6274	3.2657	3.8024	4.2521	4.6294	4.9471

Continuous Time Markov Chain Model

1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1.0000	1.3333	1.5000	1.6000	1.6667	1.7143	1.7500	1.7778
1.0000	1.5000	1.8000	2.0000	2.1429	2.2500	2.3333	2.4000
1.0000	1.6000	2.0000	2.2857	2.5000	2.6667	2.8000	2.9091
1.0000	1.6667	2.1429	2.5000	2.7778	3.0000	3.1818	3.3333
1.0000	1.7143	2.2500	2.6667	3.0000	3.2727	3.5000	3.6923
1.0000	1.7500	2.3333	2.8000	3.1818	3.5000	3.7692	4.0000
1.0000	1.7778	2.4000	2.9091	3.3333	3.6923	4.0000	4.2667

Percentage Difference

0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	11.1133	10.0018	8.5714	7.4056	6.4910	5.7671	5.1840
0.0000	10.0018	12.0922	11.8632	11.0645	10.1940	9.3794	8.6480
0.0000	8.5714	11.8982	12.7928	12.6790	12.1785	11.5519	10.9059
0.0000	7.4056	11.0904	12.6882	13.1829	13.1190	12.7844	12.3254
0.0000	6.4910	10.2119	12.1930	13.1266	13.4412	13.3985	13.1591
0.0000	5.7671	9.3899	11.5687	12.7939	13.4049	13.6218	13.5920
0.0000	5.1840	8.6549	10.9196	12.3369	13.1653	13.5957	13.7535

TABLE 2.6

Expected number of busy memories in one cycle :

Exponential distribution for tp

Constant tw=ta=E[tp]

Simulation results

	m= 2	3	4	5	6	7	8
n=2	1.4088	1.5931					
n=3	1.6185	1.9878	2.2075				
n=4		2.2198	2.5643	2.8004			
n=5			2.7980	3.1472	3.4300		
n=6				3.4088	3.7122	4.0040	
n=7					3.9990	4.3196	4.5804
n=8						4.5666	4.9028

time, with mean equal to tw.

$$\text{i.e. } \text{Prob}\{tp=x\} = \lambda \exp(-\lambda x) \quad \text{where } \lambda = 1/tw = 1/ta = 1/E[tp]$$

Note that the values in Table 2.6 lie between those predicted by Strecker and Jackson, and within 5% of the exact discrete Markov chain model for most cases. Thus, modeling the variable processing time by a constant equal to the mean processing time is a reasonable simplification. Table 2.7 shows the characteristics of the parameters in the various models.

It is important to note that with tp=tw a Pc is fast enough to make a new request to memory when the memory recovers. Thus, for a 1x1 system the memory is always busy. Also, with m≥n, if there is no contention for memory the maximum number of busy memories is min(m,n). An important result observed was the absence of a law of diminishing returns: the performance of a multiprocessor system with n processors and n memories continues to rise at a constant rate as n increases. A simple exponential server model showed this rate to be *0.5*; a constant processing time model predicted a slope of *0.586* for the average number of busy Mp's. The exponential server model gives the average number of busy Mp's as $n*m/(n+m-1)$. An approximate result for constant processing times gives the average number of busy Mp's as $i*[1-(1-1/i)^j]$, where i=max(n,m) and j=min(n,m). An intuitively obvious conclusion limits the maximum number of active Pc's by min(n,m). This maximum is reached if each processor accesses only one memory all the time.

TABLE 2.7

	Processing Time	Memory Cycle Time	Analysis	Computational Ease
Discrete Markov Chain	Constant $t_p=t_W$	Constant	Exact	Solution is algorithmic. Unwieldy for large n.
Strecker's Approximation	Constant	Constant	Approximate	Closed form solution. Simple formula.
Continous Time Markov Chain	Exponential	Exponential	Exact	Closed form solution. Simple formula.
Diffusion Approximation	Constant	Constant	Approximate	Closed form solution. Simple formula.
Simulation Model	Exponential $E[t_p]=t_W=t_a$	Constant	Approximate	Unwieldy due to slow stochastic convergence.

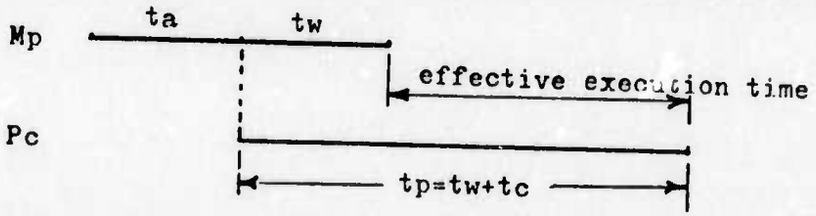
CHAPTER 3

MULTIPROCESSOR SYSTEMS WITH TP>TW

In this chapter multiprocessors with tp>tw will be discussed. First, a discrete Markov chain model for a constant processing time equal to tw+tc will be developed and a general methodology for constant tp=tw+i*tc will be presented. A more general model for processing time having a geometric distribution with its mean value greater than tw will be described. An exponential server model developed by McCredie[McCrJ73] will also be discussed.

3.1 DISCRETE MARKOV CHAIN MODELS FOR MULTIPROCESSORS WITH TP=TW+TC

In this section, discrete markov chain models will be developed for multiprocessor system in which the processing time tp is a constant and is exactly equal to tw+tc. The timing of a typical instruction is shown below.



Note that a Pc that is serviced during this cycle operates on its data during the next memory cycle. This will be modeled by associating a server with each Pc

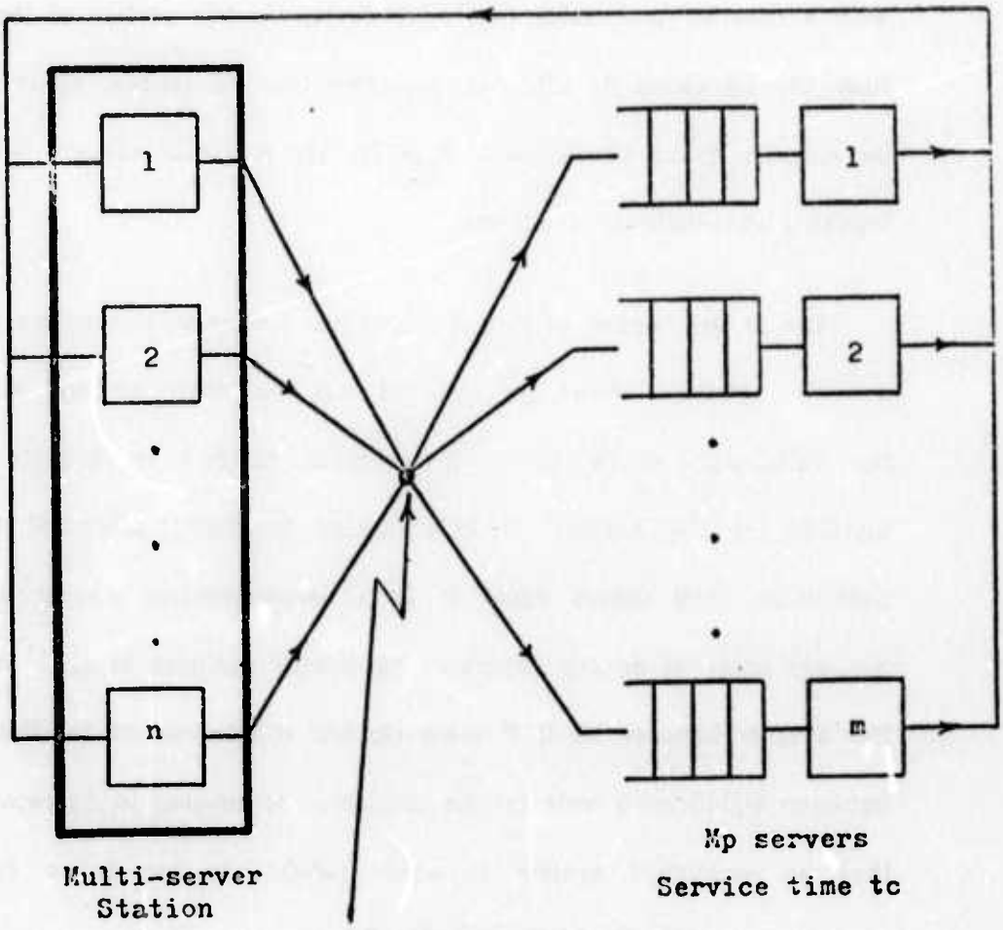


Figure 3.1 Queueing model for multiprocessors with tp=tw+tc

with a constant processing time equal to tc; and this portion of the processing time will be called its effective execution time. As before, all processors will be assumed to be identical with $P_{ij}=1/m$. The processor servers will be lumped together as a multi-server station.

Now, if the number of Pc's is n and the number Mp's is m, the state of the queueing system shown in Fig. 3.1 can be described by a (m+1)-tuple $(k_o; k_1,k_2,...,k_m)$, where k_o is the number of Pc's in execute state and $k_i, 1 \le i \le m$, is the number of Pc's queued for Mp[i]. Since all the servers contribute fixed delays equal to tc all events(entities leaving and entering queues) occur at epochs separated by integer multiples of tc. In other words, the system behaves as if it were clocked at intervals of tc. Therefore, time between significant events can be considered to advance in discrete steps. Note that an equivalent system in which tw'=0,ta'=tc and tp'=tc can also be represented by the model shown in Fig. 3.1.

Let $(k_o; k_1,k_2,...,k_m)$ be a representative state i.e. it denotes all the states that can be obtained by permuting $(k_1,k_2,...,k_m)$. Further, let $k_1,k_2,...,k_m$ be arranged in non-increasing order. Since the number of Pc's is fixed $\sum_{i=0}^{m} k_i = n$. The state space can be divided into n+1 sub-spaces corresponding to integer values of k_o ranging from 0 to n. The number of states in the 0-th sub-space is equal to the number of ways of partitioning the integer n into m parts. In general, the i-th sub-space consists of states corresponding to the partitions of n-i into m parts.

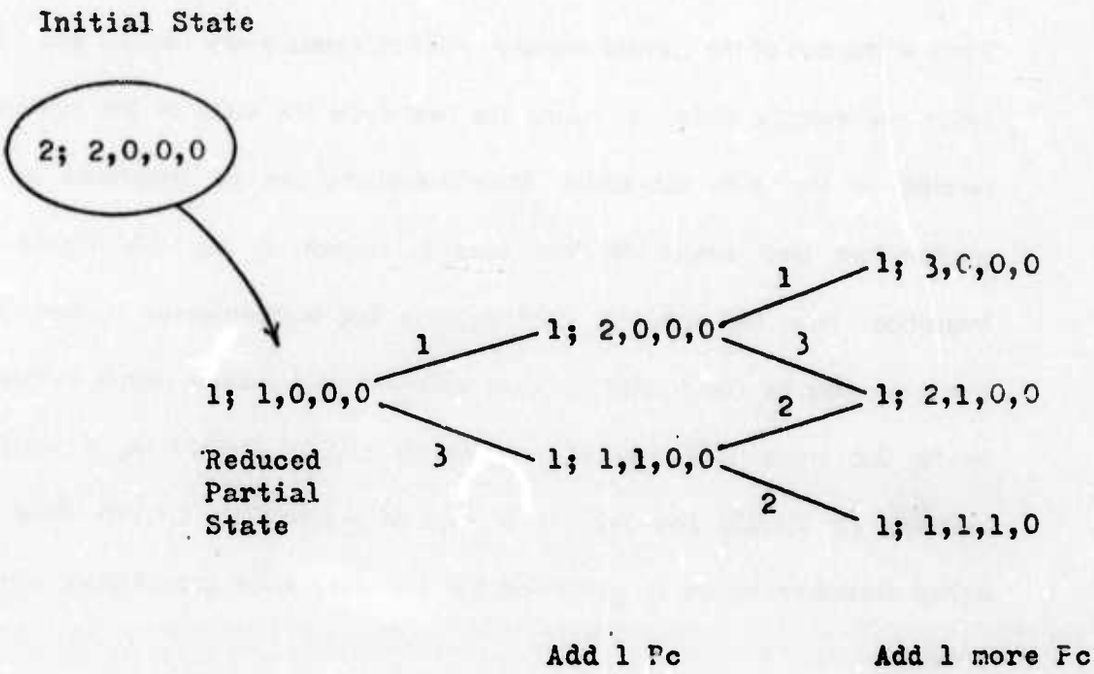


Figure 3.2 A typical enumeration tree for initial state 2; 2,0,0,0

Chapter 3 : Multiprocessors with tp>tw

3.1 Discrete Markov Chain Model for tp=tw+tc

Consider a state vector in the i-th sub-space. The value of k_0 is i. Let d denote the number of non-zero parts of the memory-state-vector $(k_1,k_2,...,k_m)$. Then at the end of the current memory cycle i Pc's make a new request and d Pc's enter the execute state. i.e. during the next cycle the state of the system is located in the d-th sub-space. State transitions can be described by an enumeration tree similar to that used in chapter 2, Fig. 2.3. Figure 3.2 transitions from the state (2; 2,0,0,0). for a 4x4 multiprocessor system. Such trees can now be constructed for each state and the transition matrix evaluated. In fig 3.2, there is 1 way of reaching (1; 3,0,0,0), (3+3*2) i.e. 9 ways of reaching (1; 2,1,0,0) and 3*2 i.e. 6 ways of reaching (1; 1,1,1,0). Once the entire transition matrix is generated the stationary state probabilities can be obtained.

The number of states of this discrete Markov chain model increases faster than the exact Markov chain model for tp=tw, described in chapter 2. Table 3.1 compares the number of states for the two models. An approximate Markov chain model will now be proposed. The system behavior will be modified to simplify the analysis. At the end of each cycle the active Pc's (those that are served by memory during the current cycle) will enter the execute state. However, all the queued Pc's will be removed from the memory queues and will be allowed to make new requests to memory along with those Pc's that were in execute state during the current cycle. Let the current state be $(k_0; k_1,k_2,...,k_m)$, with exactly d Mp queues occupied. Then, at the end of the current cycle, d Pc's enter the

TABLE 3.1

Comparison of the Number of States for

Discrete Markov Chain Models for

tp=tw and tp=tw+tc

n=m	Number of States tp=tw	tp=tw+tc
1	1	1
2	2	3
3	3	6
4	5	11
5	7	18
6	11	29
7	15	44
8	22	66
9	30	96
10	42	138
11	56	194
12	77	271

execute state and n-d Pc's make new requests. Thus, for this simplified model, the state of the system is characterized by the number of Pc's that make new requests at the beginning of a cycle.

Now, if i Pc's make a new request to m Mp's, the probability that Mp[i] gets at least one request is $1-(1-1/m)^i$. With i Pc's accessing m Mp's simultaneously, the number of busy Mp's can take any integer value from 1 upto min(i,m). The probability of exactly j Mp's being occupied is given by the ratio of the number of ways that j Mp's can be occupied and i-j Mp's not be occupied to the total number of ways of assigning i Pc's among m Mp's i.e. m^i. In section 2.3, it was stated that the number of ways that exactly j out of m Mp's can be occupied by i Pc's is CM(m,i)*S(i,j). Now, if j Mp's are occupied during the current cycle then n-j Pc's make a request to Mp during the next cycle. Therefore, given that i Pc's made a request to Mp at the beginning of the current cycle, the conditional probability that n-j Pc's will make a new request at the beginning of the next cycle (which is also the end of the current cycle) is

$$CM(m,i)*S(i,j)/m^i$$

Note that the above expression denotes the probability of a transition from a current state i to a next state n-j. The value of j can range from 1 to min(i,m).

For a multiprocessor system with n Pc's and m Mp's, let k=min(m,n). The number of occupied memories in a cycle ranges from 0 to k. Therefore, the number

TABLE 3.2

Transition matrix for a 4x4 system with tp=tw+tc

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.09375 \\ 0.0000 & 0.0000 & 0.0000 & 0.3750 & 0.56250 \\ 0.0000 & 0.0000 & 0.7500 & 0.5625 & 0.32813 \\ 0.0000 & 1.0000 & 0.2500 & 0.0625 & 0.01563 \\ 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.00000 \end{bmatrix} \times \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}$$

$$X_0 + X_1 + X_2 + X_3 + X_4 = 1$$

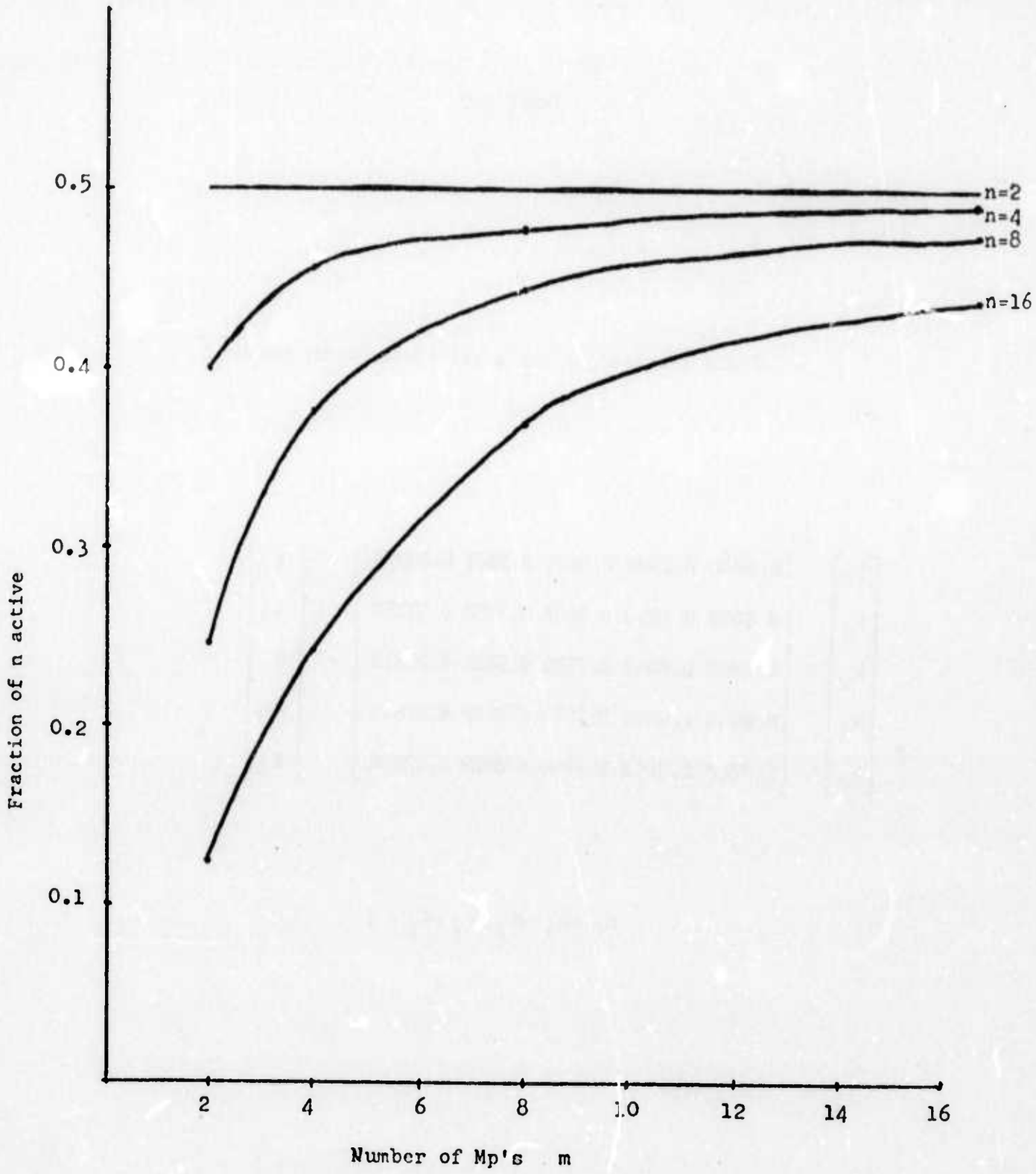


Figure 3.3a Execution Rate as a fraction of the number of Pc's

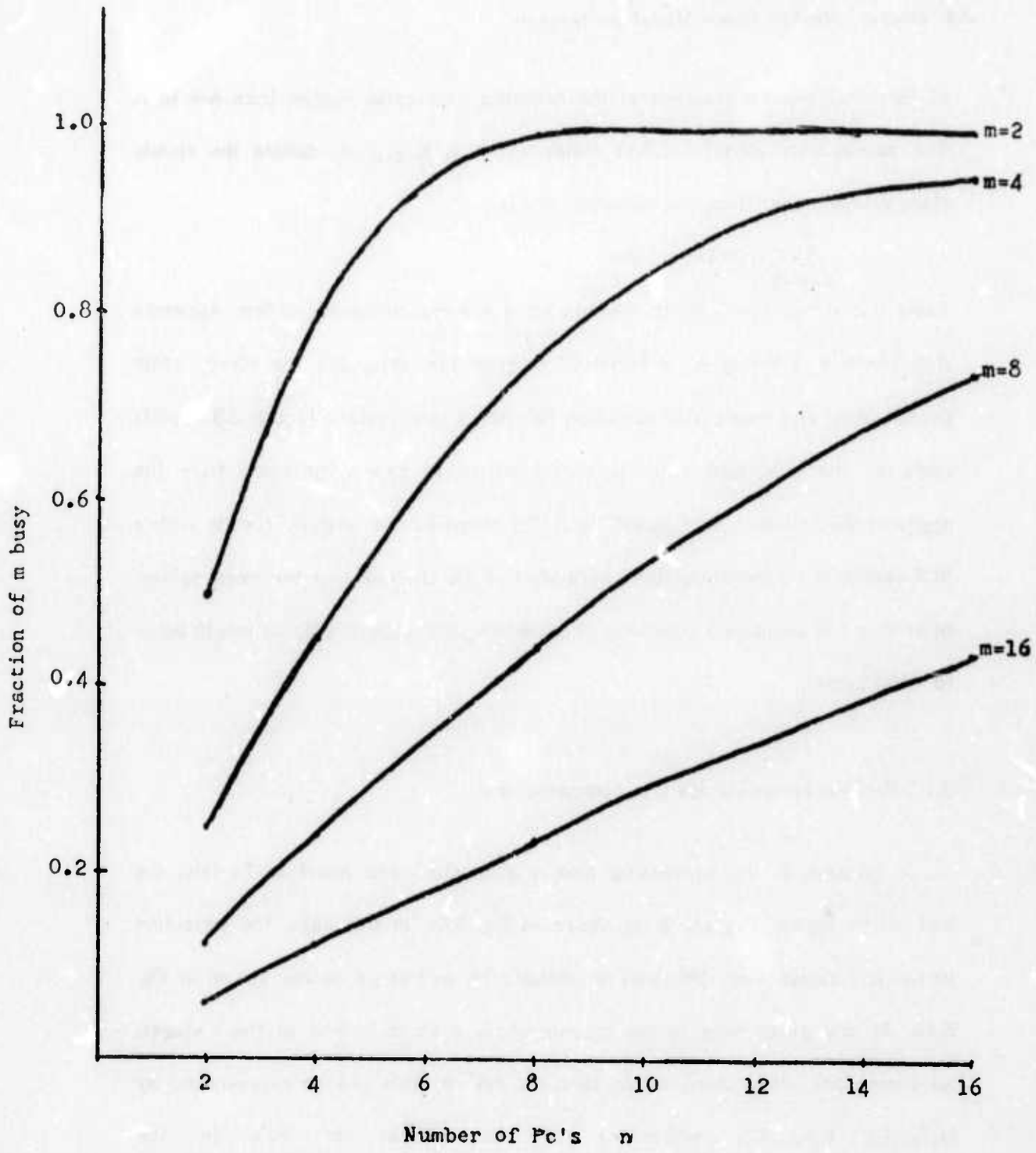


Figure 3.3b Execution rate as a fraction of the number of Mp's

of Pc's that can be assigned at the beginning of a cycle ranges from n-k to n. The approximate model has k+1 states. Let X_{n-k}, X_{n-k+1},...,X_n denote the steady state probabilities. Then, the execution rate is

$$\sum_{i=n-k}^{n} X_i*[1-(1-1/m)^i]*m$$

Table 3.2 shows the transition matrix for a 4x4 multiprocessor sytem. Appendix A-5 contains a listing of a FORTRAN program that computes the steady state probabilities and hence the execution rate for a nxm system. Figure 3.3 depicts plots of the execution rate as a function of m and n, obtained from the approximate Markov chain model. Table 3.3 compares the analytic results with a 90% confidence interval obtained by a Monte Carlo simulation of the exact system behavior. The simulation consisted of 10 independent experiments of length equal to 4000 cycles.

3.1.1 General Technique for Constant tp=tw+i*tc

In general, if the processing time is a constant and equal to tw+i*tc, the instruction timing diagram is as shown in Fig. 3.4a. In this case, the execution phase is i cycles long. This can be modeled by an i-stage server shown in Fig. 3.4b. At any given time in the execute phase a Pc is in one of the i stages; advancing one stage every cycle. Now, the system state can be represented by $(j_1,j_2,...,j_i;\ k_1,k_2,...,k_m)$, where j_l is the number of Pc's in the execute-stage l, and the k_l's denote the memory queue sizes. As before, let d denote the number of non-zero k_l's. Then, at the beginning of the next cycle, d

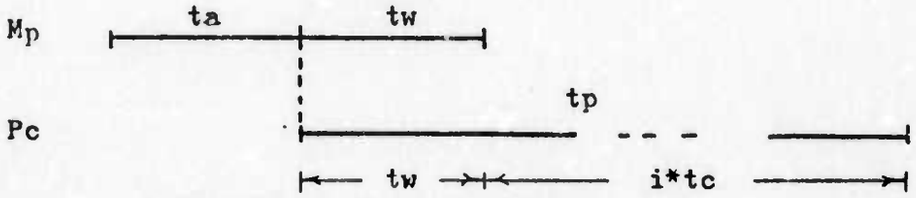


Figure 3.4a Instruction timing diagram for tp=tw+i*tc

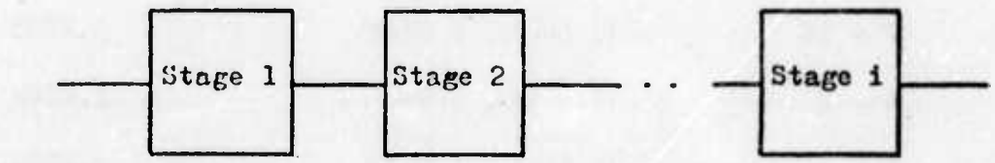


Figure 3.4b i-stage server model for a Pc

TABLE 3.3

Average Number of Busy Mp's

tp=tw+tc

n × m	90% confidence interval from simulation	Analytic Result
2 × 2	(1.0000 , 1.0000)	1.0000
2 × 4	(1.0000 , 1.0000)	1.0000
2 × 8	(1.0000 , 1.0000)	1.0000
2 × 16	(1.0000 , 1.0000)	1.0000
4 × 2	(1.5647 , 1.5843)	1.6000
4 × 4	(1.8194 , 1.8276)	1.8276
4 × 8	(1.9126 , 1.9244)	1.9197
4 × 16	(1.9418 , 1.9817)	1.9612
8 × 2	(1.8251 , 1.8547)	1.9692
8 × 4	(2.8460 , 2.9206)	3.0259
8 × 8	(3.4858 , 3.5346)	3.5530
16 × 2	(1.9086 , 1.9404)	1.9998
16 × 4	(3.4936 , 3.5677)	3.8772
16 × 8	(5.4431 , 5.6254)	5.9053
16 × 16	(6.8140 , 6.9788)	7.0136

Pc's enter execute-stage 1; j_1 Pc's make new requests to Mp and rc's in the other execute stages advance to the next higher execute stage. Thus, for example, if the current state is(2,1,0,4; 2,1,1,0) then the next partial state is (3,2,1,0; 1,0,0,0) and 4 Pc's make new requests. The different ways that these Pc's can be assigned determine the k_i's in the next state, which can be obtained by using an enumeration tree similar to the one described earlier.

3.2 DISCRETE MARKOV CHAIN MODELS FOR GEOMETRICALLY DISTRIBUTED TP

For our next model, let the processing time have a geometric distribution given by,

$$\text{Prob}[tp=tw+i*tc] = \beta*\alpha^i$$

where $\beta=1-\alpha$

Then, the mean processing time is given by,

$$\sum_{i=0}^{\infty} \beta*\alpha^i*(tw+i*tc)$$

$$\text{i.e. } tw*\sum_{i=0}^{\infty}\beta*\alpha^i + \beta*tc*\sum_{i=0}^{\infty} i*\alpha^i$$

$$\text{i.e. } tw + tc*\beta*\sum_{i=0}^{\infty} i*\alpha^i$$

$$\text{i.e. } tw + tc*\beta*\alpha/(1-\alpha)^2$$

$$\text{i.e. } tw + \alpha*tc/(1-\alpha)$$

Thus any mean value of tp greater than tw can be modeled by appropriately choosing α. This model is more general than the models of the previous section.

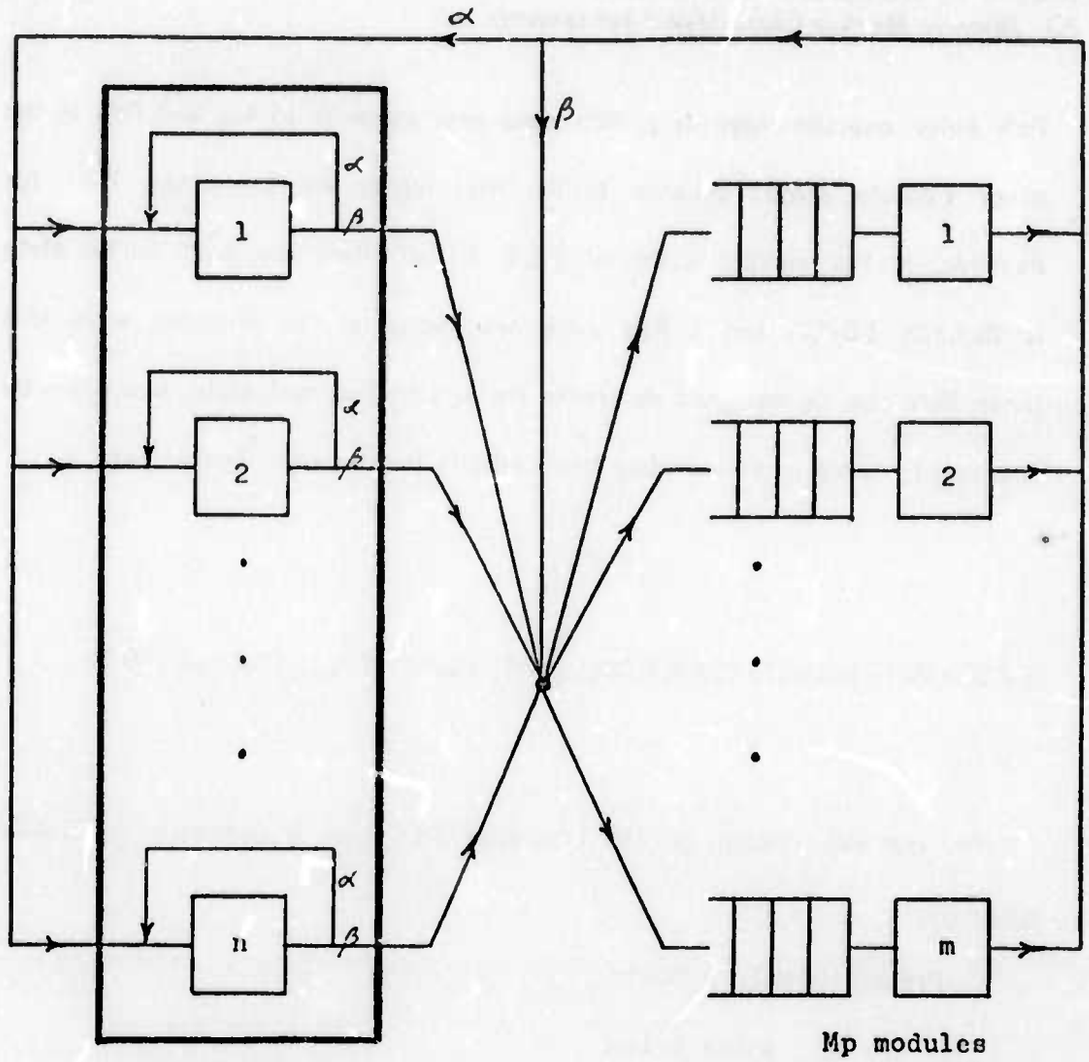


Figure 3.5 Structure of the queueing model for geometrically distributed processing time.

Also, a single model with α as a parameter handles all cases where the mean value of tp is greater than tw. The geometric distribution is a discrete analog of the exponential distribution. The measurements reported in chapter 5 show that the processing time distribution is indeed close to a shifted exponential.

Once again, the representative state of an exact discrete Markov chain model is given by the vector $(k_c,k_1,k_2,\ldots,k_m)$. Also, $\sum_{i=0}^{m} k_i=n$ and k_i is the number of Pc's queued for Mp[i], $1\le i\le m$, and k_c is the number of Pc's in execute state. The reduced partial state at the end of the current cycle is given by the vector $(0,j_1,j_2,\ldots,j_m)$, where j_i max$(0,k_i-1)$ for $1\le i\le m$. Note that $\sum_{i=1}^{m} j_i = n-k_0-d$, where d is the number of non-zero k_i's. Now, upto k_0+d Pc's are potentially available to be reassigned to the various queues. Let the next state be denoted by $(l_0,l_1,l_2,\ldots,l_m)$. Figure 3.5 shows the structure of the queueing model. Each Pc has a probability α of going back into the execute phase. Since l_0 denotes the number of Pc's that are in the execute phase during the next cycle, the range of l_0 is from 0 to k_0+d. The value of l_0 is governed by the following probability function,

$$\text{Prob}\{l_0=i\} = \binom{k_0+d}{i} * \alpha^i * \beta^{k_0+d-i} \quad 0\le i\le k_0+d$$

The rest of the next state vector can be determined by using an enumeration tree. Figure 3.6 shows a typical enumeration tree for an initial state of (2, 2,0,0,0).

Once again, to reduce the number of states let us make the following approximation. At the end of an Mp cycle, those Pc's that were in the Mp queues

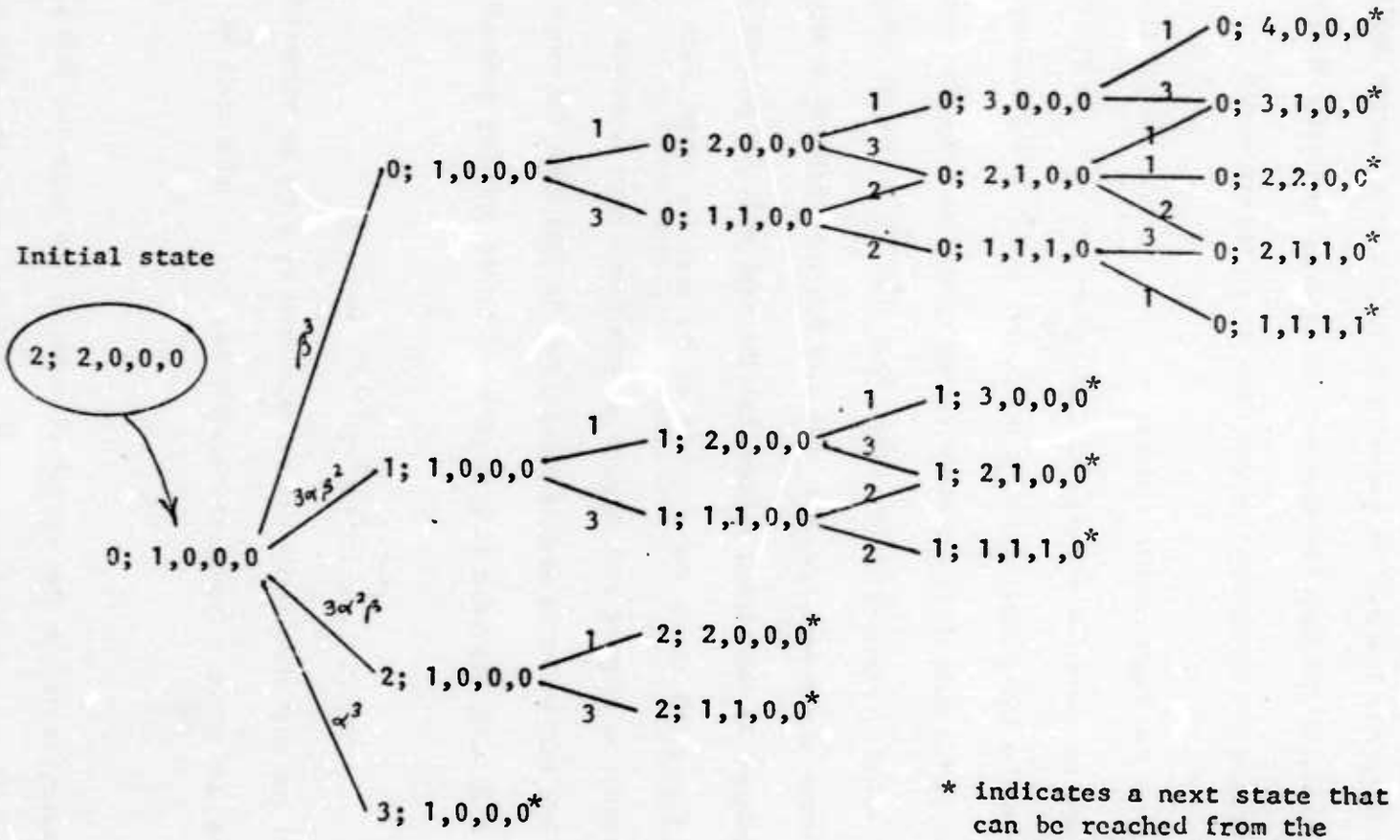


Figure 3.6 A typical enumeration tree

during the current cycle but not serviced are removed from the queues. They are then reassigned to the Mp queues. Now, the state is characterized by the number of Pc's that make a request i.e. the number of Pc's that are queued during the cycle. Thus, the number of states is n+1, viz. 0,1,2,...,n. Let the number of Pc's queued during the current cycle be i. This means that the other n-i Pc's are executing. If the i requests at the beginning of the current cycle result in d(≤min(i,m)) busy Mp's during the current cycle, then i-d Pc's are left unserviced. Therefore, during the next cycle at least i-d Pc's are queued for Mp service. Besides, each of the other n-i+d Pc's has a probability β of making a request to memory.

The probability that i-d Pc's are left in the Mp queues at the end of the current cycle is given by

$$CM(m,d)*S(i,d)/m^i$$

The above expression ensues from the fact that i Pc's make random requests to m Mp's resulting in exactly d Mp's being occupied. Also, the probability that j out of the other n-i+d Pc's make a request at the beginning of the next cycle is

$$\binom{n-i+d}{j} * \beta^j * \alpha^{n-i+d-j}$$

Therefore, the probability that i-d+j Pc's make a request to Mp during the next cycle, i.e. probability that the next state is i-d+j is

$$CM(m,d)*S(i,d)/m^i * \binom{n-i+d}{j} * \beta^j * \alpha^{n-i+d-j}$$

Table 3.4 summarizes some of the numerical results obtained from the approximate model. A listing of the FORTRAN program for this model is included in Appendix

TABLE 3.4

Average number of busy Mp's
tp -- Geometric Distribution

n × m	α	90% confidence interval from simulation	Analytic Result
4 × 2	0.1		1.8478
	0.25		1.7849
	0.5		1.5423
	0.75	(0.9130 , 0.9587)	0.9408
4 × 4	0.1		2.6079
	0.25	(2.2875 , 2.3258)	2.3679
	0.5	(1.7382 , 1.8022)	1.7920
	0.75		0.9739
4 × 8	0.1		3.0773
	0.25	(2.6607 , 2.6831)	2.6824
	0.5		1.9012
	0.75		0.9877
8 × 4	0.1		3.5442
	0.25		3.4213
	0.5	(2.7675 , 2.8254)	2.9809
	0.75		1.8620
8 × 8	0.1		5.0243
	0.25	(4.3508 , 4.4713)	4.5904
	0.5	(3.4158 , 3.5118)	3.5224
	0.75		1.9390
16 × 8	0.1		6.9461
	0.25		6.7058
	0.5	(5.3855 , 5.5805)	5.8650
	0.75		3.7050
16 × 16	0.1		9.8718
	0.25		9.0441
	0.5	(6.7643 , 6.9240)	6.9848
	0.75		3.8693

A-6. This model will be used in chapter 5 to predict the performance of C.mmp. For tp>tw this model is fairly realistic: the processing time is modeled as a geometrically distributed random variable and the memory cycle time is a constant.

3.2.1 Extensions

The basic geometric model can be used to model systems in which each Pc has a private cache memory (Mc). Let the access time of the cache be tf (≤tc); and let x denote the probability of making an access to the cache. Therefore, a fraction (1-x) of all memory requests is to Mp. In the queueing model shown in Fig. 3.7, α denotes the probability that the execution phase goes through another cycle. Thus, $\beta = 1-\alpha$ is the probability of a Pc finishing execution and making a request to memory. Thus, $\beta_1=\beta*(1-x)$ and $\beta_2=\beta*x$.

As before,

$$\text{Prob}\{tp=tw+i*tc\} = \beta_1*\alpha^i \qquad \text{for } i=0,1,2,...$$

and,

$$\text{Prob}\{tp=tc-tf+i*tc\} = \beta_2*\alpha^i \qquad \text{for } i=0,1,2,...$$

The expected value of tp, given that all accesses are to Mp, is given by

$$E[tp \mid Mp] = tw + tc*\beta_1*\alpha/(1-\alpha)^2$$

Similarly,

$$E[tp \mid Mc] = tc-tf + tc*\beta_2*\alpha/(1-\alpha)^2$$

Hence, the unconditional expected value of tp is

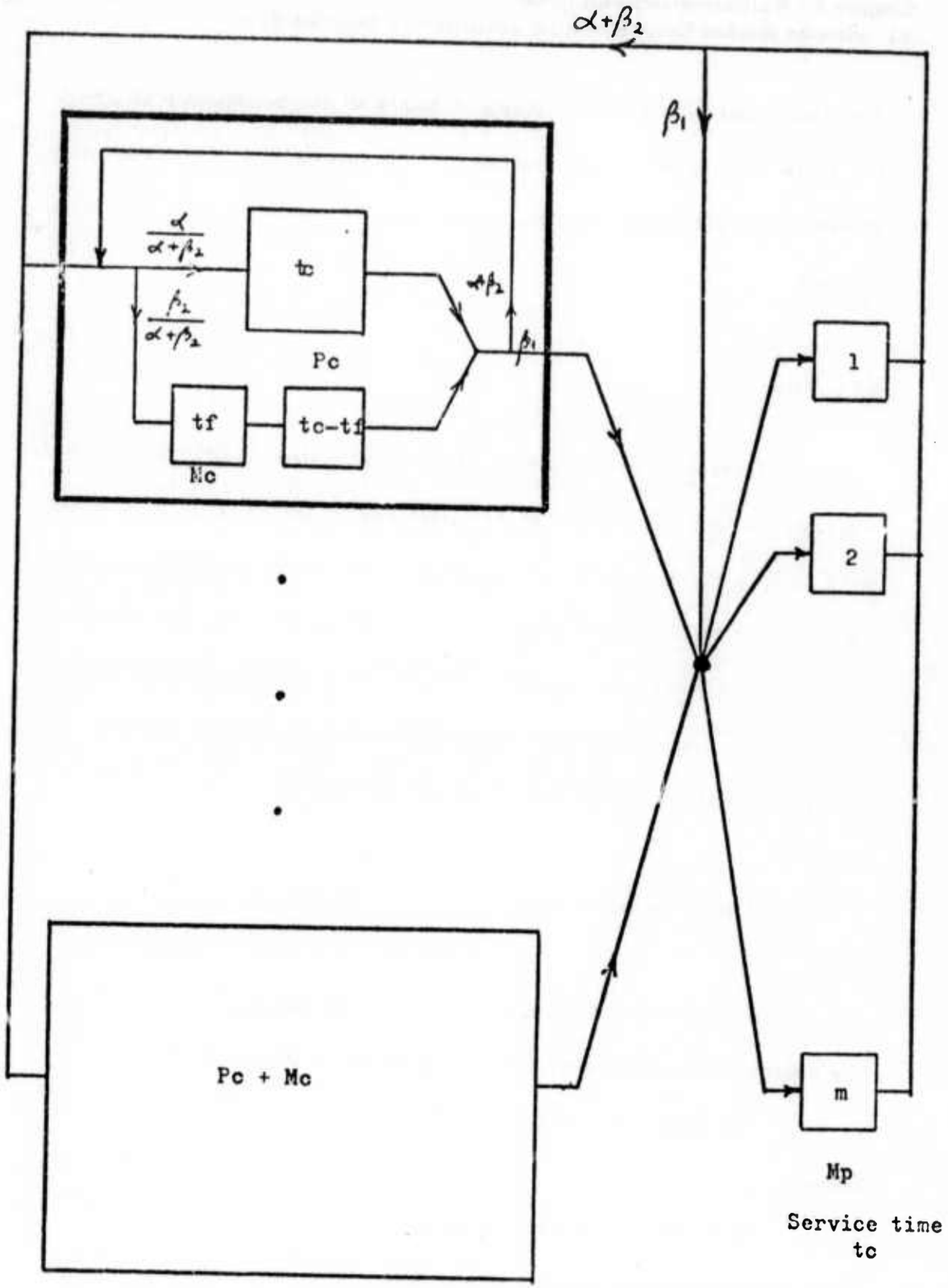


Figure 3.7 Queueing model for multiprocessors with cache memory

$$E[tp] = (1-x)*E[tp : Mp] + x*E[tp : Mc]$$

$$= (1-x)*[tw + tc*(1-x)*\alpha/(1-\alpha)] + x*[tc-tf + tc*x*\alpha/(1-\alpha)]$$

If E[tp] is known, α can be computed from the above equation. This model is fairly general in that the parameters x and E[tp], tc, tw and tf can be chosen arbitrarily, subject to $E[tp] \geq tw+x*(tc-tf-tw)$ and $tf \leq tc$. The distribution of tp is now the sum of two geometrics.

Note that this model, as viewed by the Mp, behaves exactly like the model for geometrically distributed tp shown in Fig. 3.5. The β in Fig. 3.5 is equivalent to the β_1 in fig 3.7, and the α in Fig. 3.5 is equivalent to the $\alpha+\beta_2$ in fig 3.7. Let R denote the average number of busy Mp's for the model of Fig. 3.5, but with $\beta=\beta_1$, and $\alpha=1-\beta_1$. The values of tw and tc is kept unchanged. Now, in the system with the cache, the average number of busy Mp's is also R. Note that the expected value of tp is different for the two cases. Now, by definition, for every 1-x accesses to Mp there are x accesses to Mc. Consider an interval equal to T Mp cycles. During this interval there are R*T busy Mp cycles. Hence, there are R*T*x/(1-x) accesses to Mc. Therefore, the number of instructions executed in one Mp cycle is

$$R + R*x/(1-x)$$

$$\text{i.e. } R/(1-x)$$

Thus, the average number of equivalent Mp cycles is R/(1-x).

For example, let us compute the average number of effective busy Mp cycles for ta=tw=5, tc=10, tf=1, x=0.75, E[tp]=15, and n=m=8. The equation for E[tp] in

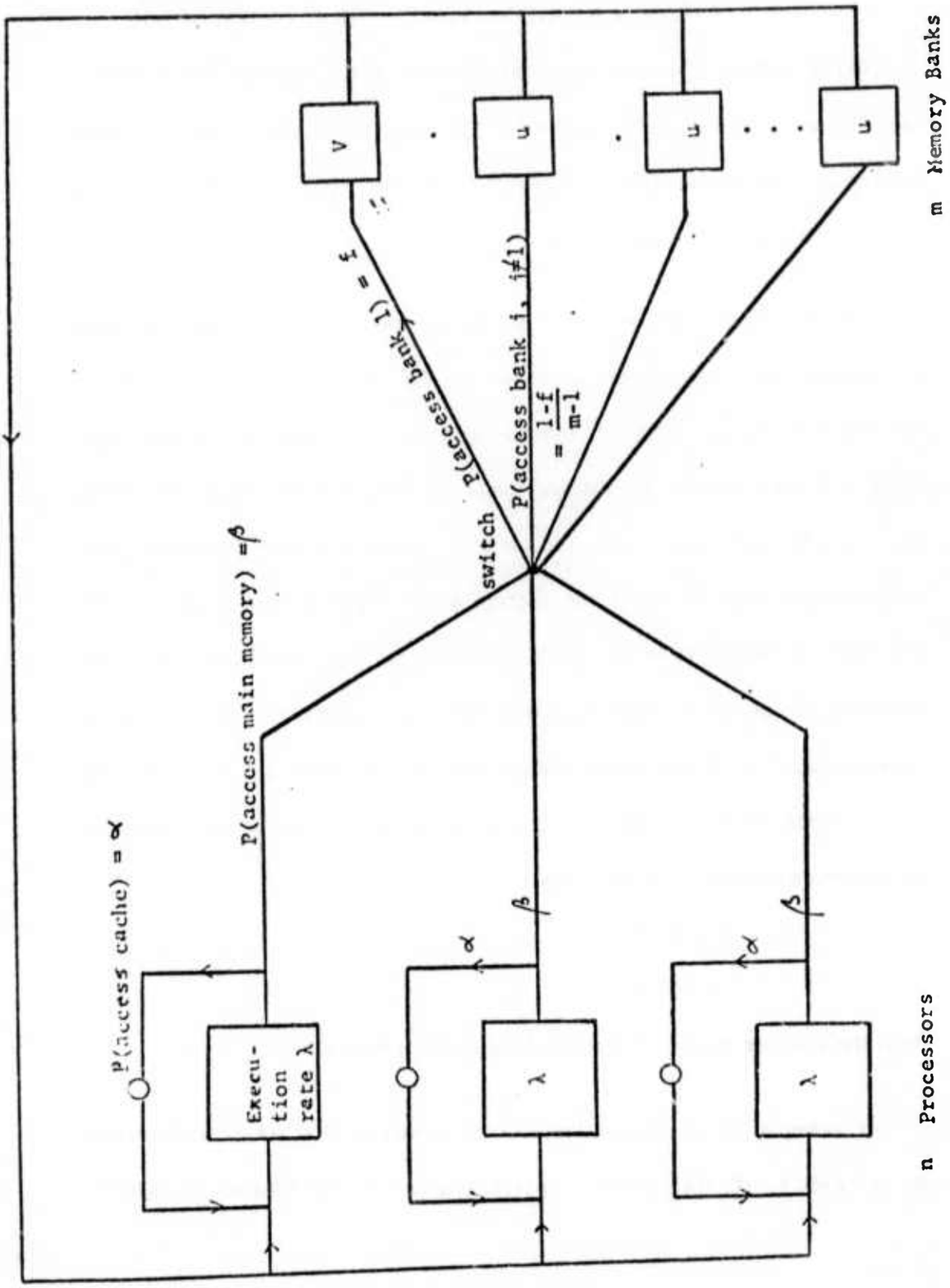


Figure 3.8 Structure of McCredie's Memory Interference Model

terms of α gives α=28/53. Hence β_1=(1-x)*(1-α)=0.25*25/53. Using β=0.25*25/53 for the model of section 3.2, we get R=0.9371. Hence, the average number of effective busy Mp cycles is 0.9371/0.25 = 3.7484.

3.3 McCREDIE's EXPONENTIAL SERVER MODEL

As mentioned in Chapter 2, if the memory cycle time and the processing time are assumed to be exponential, the queueing results of Jackson[JackJ63] can be used. The structure of McCredie's memory interference model[McCr73] is depicted in Fig. 3.8. In terms of the notation used in this chapter, λ=1/(tp-tw) (if α=0),u=1/tc and α is the probability of accessing a cache memory whose access time is assumed to be negligible. The time from the completion of one reference to main memory until the next access is exponentially distributed with mean $1/(\lambda*\beta)$. The model allows the first Mp module to have a different cycle time 1/v. The probability that a request to main memory is to the first Mp module is f; the requests to the other Mp modules being uniformly distributed. Let k be the number of Pc's queued. Using Jackson's formulae and some clever grouping of terms† the execution rate is obtained as

$$\sum_{k=0}^{n} (n-k)*\lambda*P(k)$$

---

†The reader is referred to McCr73 for the details of the derivation.

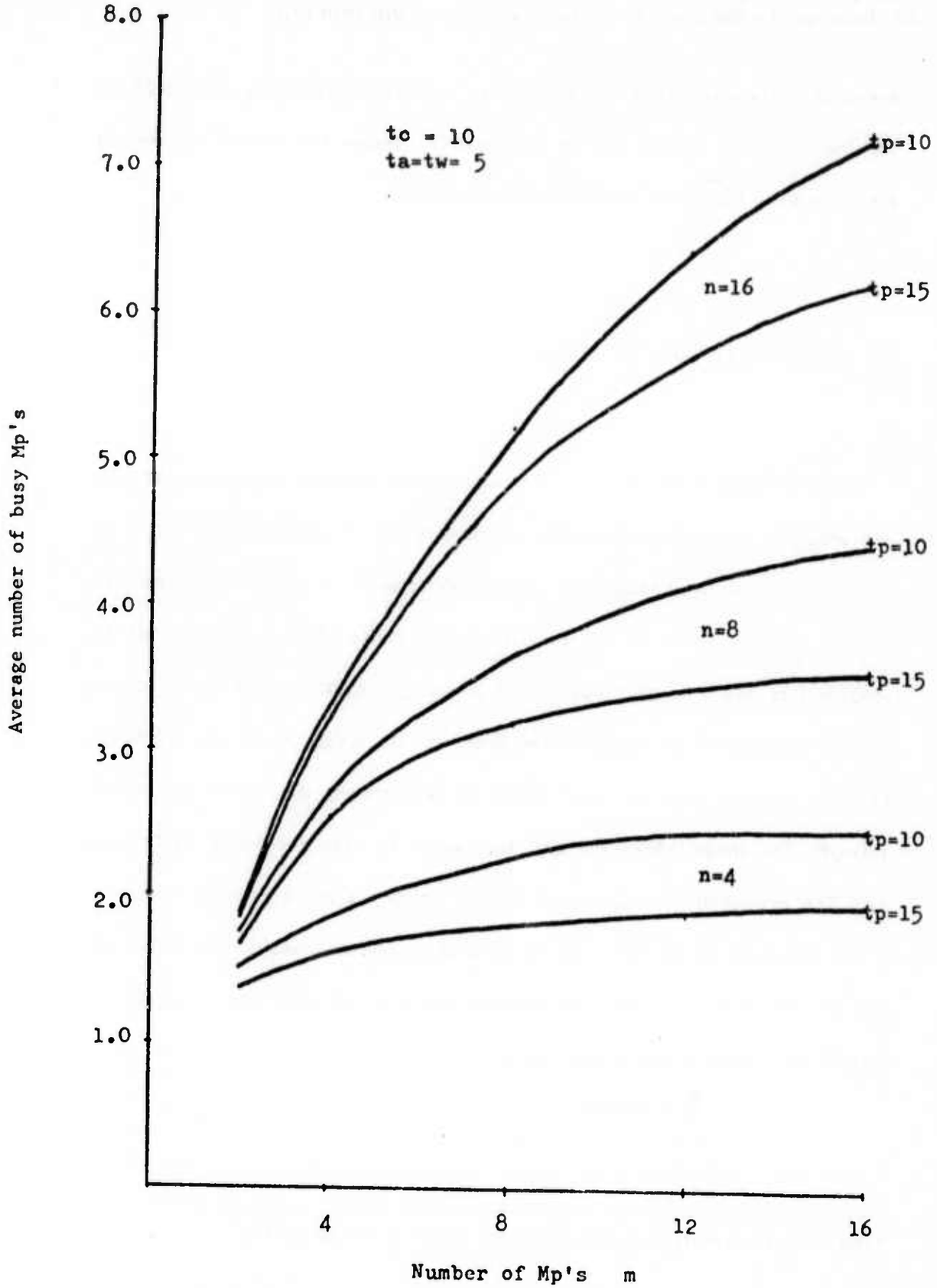


Figure 3.9 Performance predicted by McCredie's model

$$\text{where } P(k) = \frac{n!\,(\beta\lambda)^k \sum_{i=0}^{k} \binom{m-2+i}{i} \left(\frac{f}{v}\right)^{k-i} \left(\frac{1-f}{(m-1)u}\right)^{i}}{(n-k)! \sum_{k=0}^{n} W(k).T(k)}$$

$$\text{and } W(k) = (\beta*\lambda)^k * n!/(n-k)!$$

$$\text{and } T(k) = \sum_{j=0}^{k} \binom{m-2+j}{j} \left(\frac{f}{v}\right)^{k-j} \left(\frac{1-f}{(m-1)u}\right)^{j}$$

Figure 3.9 shows some of the results predicted by the model. The advantage of this model is that it allows a cache memory and an Mp module with a different speed and a different access probability. Note that with the presence of the cache $\lambda \neq 1/(tp-tw)$; but $\lambda = 1/(tp-\beta*tw)$.

3.4 STRECKER'S ANALYSIS

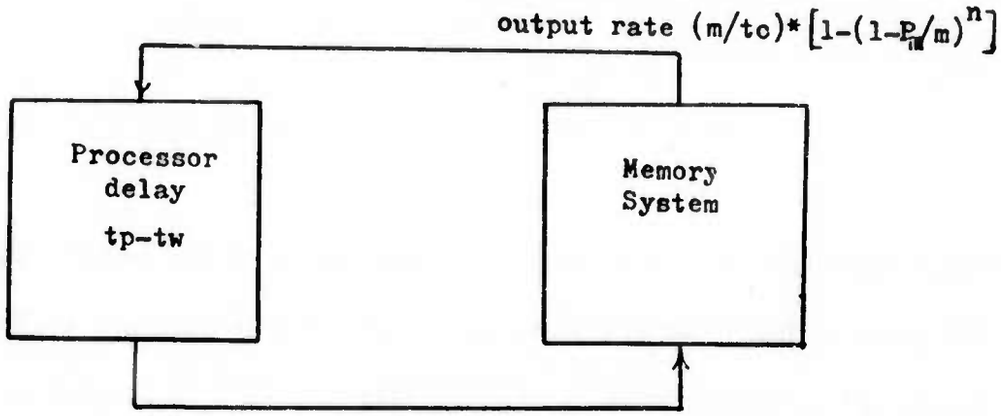
This section will briefly review Strecker's analysis of multiprocessors with tp>tw[StreW70]. The processing time is assumed to be a constant. The number of processors queued is, in general, less than n. The probability a Pc is queued is denoted by p_m. Then, assuming a binomial distribution for the queued processors, the execution rate is

$$m*[1-(1-p_m/m)^n]$$

Now, because the number of Pc's queued is binomially distributed, the average number of Pc's queued is $n*p_m$. Hence,

p_m = average number of Pc's queued/n

Strecker's flow diagram of the instruction execution is shown below.



Strecker states that the average number of Pc's not queued is the product of the average unit execution rate and the effective Pc delay tp-tw†. Thus,

$$p_m = 1 - (m/n/tc) * (1-(1-p_m/m)^n) * (tp - tw)$$

which is a n-th order polynomial equation in p_m, which has only one solution for p_m in the interval (0,1). This value of p_m is used in the earlier expression for the execution rate. This model is fairly simple and good for larger values of tp.

---

†This follows directly from Little's formula, L=λ*W [cf. LittJ61].

3.5 THE EFFECT OF CACHE ON SYSTEMS WITH tp≥tw

One of the techniques used to increase the execution rate of uniprocessor systems involves the use of a fast cache memory, Mc. In multiprocessor systems, the cache not only provides a memory with a smaller access time but also reduces the traffic through the crosspoint switch. This reduces the memory interference and reduces the amount of time the processor spends waiting for Mp service. In fact, the use of private caches for the Pc's is being considered for CMU's C.mmp.

In this section, we shall characterize the processing time by a single constant value tp; with $P_{ij}=1/m$. Let the cache have an access time tf (≤ta) and a rewrite time tr (≤tw). Since tp≥tw and tr≤tw, the cache always recovers before the Pc can make its next request. Let α be the probability of accessing Mc i.e. the probability that the cache contains the information needed by the Pc. The probability of accessing Mp is $\beta=1-\alpha$.

Now, the time needed to execute one unit instruction out of cache is Wc=tp+tf. Let Wm be the average time needed to execute one unit instruction out of Mp. Hence, the average time needed to execute one unit instruction is

$$Wavg = \alpha * Wc + \beta * Wm$$

Therefore, the execution rate is n/Wavg

The problem now reduces to evaluating Wm. Let us focus our attention on a

single Mp unit. Further, let us assume that the number of queued Pc's for that Mp unit follows the binomial distribution; and let p_m be probability that a Pc is queued for one of the Mp's. Hence, the probability of being queued for the Mp unit under consideration is p_m/m, since all Mp's have the same speed and $P_{ij}=1/m$. From the binomial distribution for the queued Pc's, it follows that, L, the average number of Pc's queued for the Mp is $n*p_m/m$. The rate λ at which Pc's are served by this Mp is $m*[1-(1-p_m/m)^n]/tc$. Using Little's formula, $L=\lambda*W$, we obtain the average waiting time for an Mp as

$$n*p_m/m * tc / [1-(1-p_m/m)^n]$$

Therefore Wm, the average time for one instruction out of Mp, is

$$tp-tw + n*p_m/m * tc / [1-(1-p_m/m)^n]$$

Hence,

$$Wavg = \alpha*(tp-tw) + \beta*\{n*p_m/m * tc / [1-(1-p_m/m)^n]\}$$

Now, the only undetermined quantity is p_m. In T Mp cycles the total number of busy Mp cycles is

$$T_1 = m * [1-(1-p_m/m)^n] * T$$

Hence, the total number of busy cache cycles is

$$T_2 = \alpha/\beta *\{m * [1-(1-p_m/m)^n] * T\}$$

and the total number of unit instructions is the sum of the above two expressions, i.e. T_1+T_2. Therefore, the unit execution rate is

$$m/(\beta*tc) * [1-(1-p_m/m)^n]$$

Recall that we had earlier found the execution rate to be n/Wavg. Equating the two and rearranging the terms we get

$$p_m = 1 - m/(n*\beta*tc) * [\alpha*(tp+tf) + \beta*(tp-tw)] * [1-(1-p_m/m)^n]$$

The above equation can be solved iteratively for p. A FORTRAN program that computes the execution rate for this model is listed in Appendix A-7. Figure 3.10 shows the execution rate of an 8x8 and a 16x16 multiprocessor system for varios values of the other parameters. Some simulation confidence intervals are also depicted. Note that with α=0, this model yields the same results as Strecker's model described in Section 3.4. The major advantage of this model is that it allows the cache speed to be a control parameter of the analysis.

3.6 CONCLUDING REMARKS

In general, if the processing time is greater than the memory rewrite time, then in the absence of memory contention one instruction is executed by each Pc every ta+tp time units. Thus, the theoretical maximum value of the average number of busy Mp's is n*tc/(ta+tp). Hence, if m≥n, even in the absence of memory interference, the Mp's will have idle periods. Comparing Fig. 2.11 and Fig. 3.3a, the curves for tp>tw reach their asymptotic maximum for smaller values of m. Systems with tp>tw are generally processor speed limited and relatively small performance improvement will be obtained if the memory speed is increased.

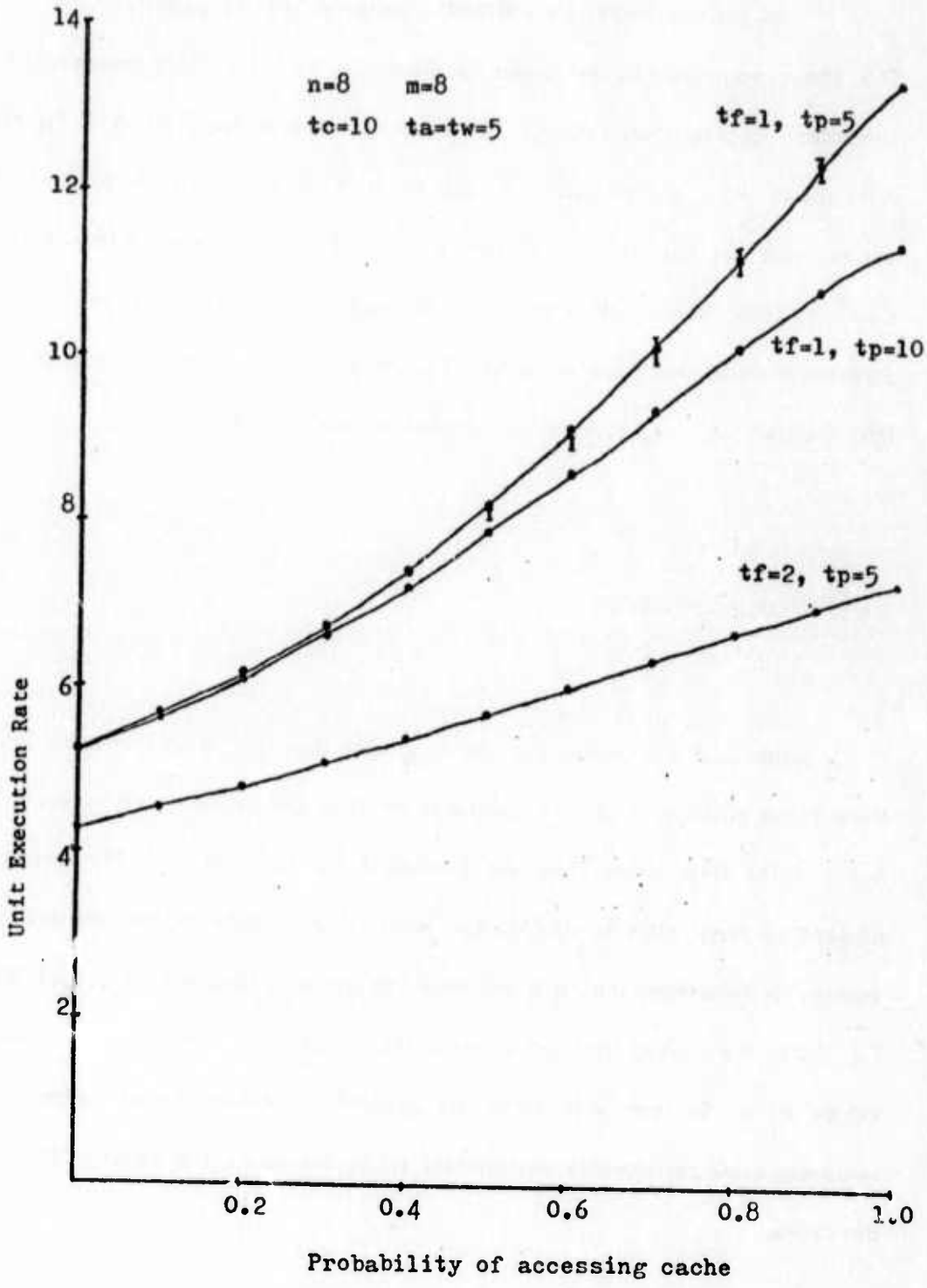


Figure 3.10a The Effect of Cache on a 8x8 system

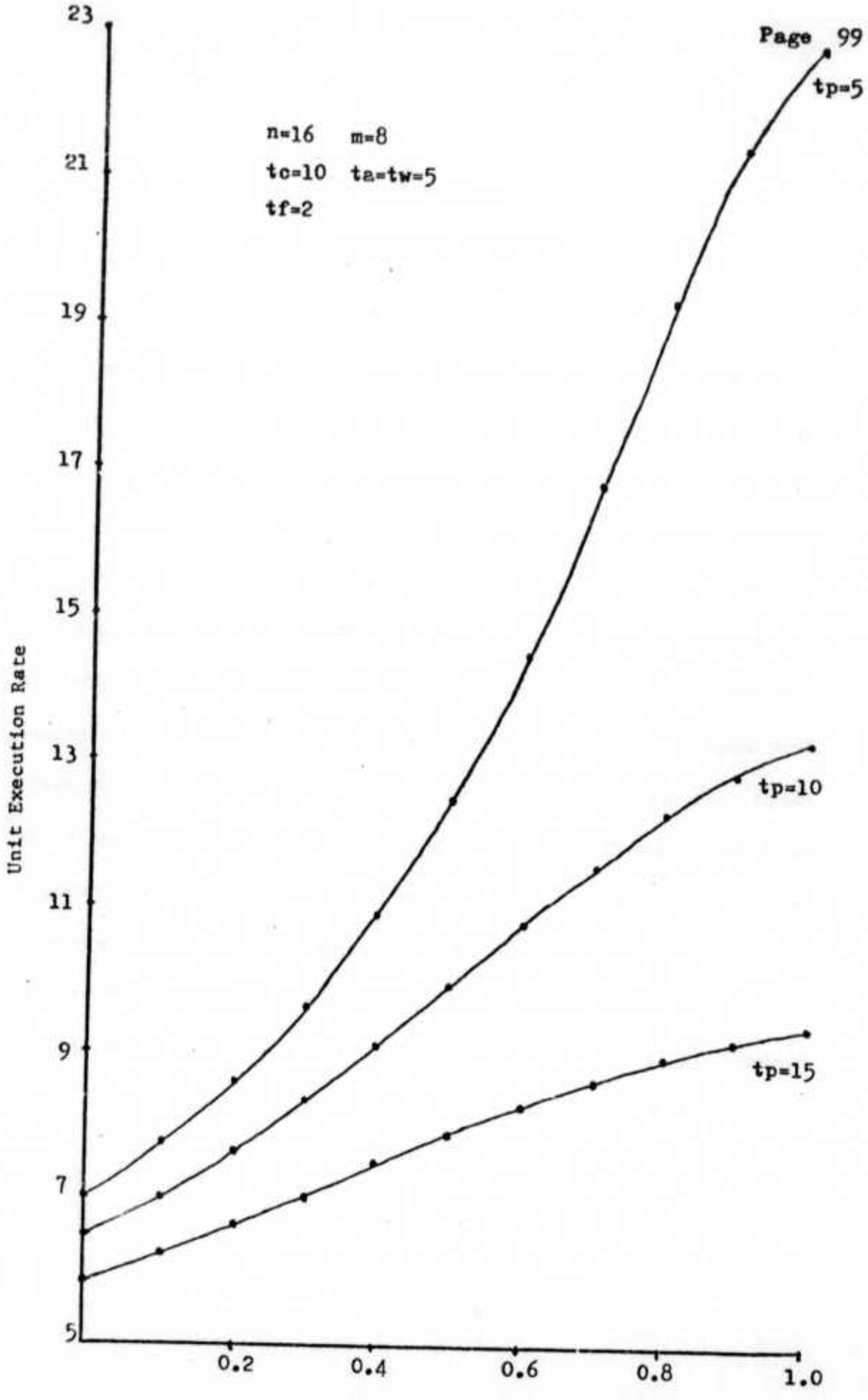


Figure 3.10 The Effect of Cache on 16x8 systems

CHAPTER 4

MULTIPROCESSOR SYSTEMS WITH TP<TW

In this chapter, an approximate model is proposed for multiprocessor systems with tp<tw and $P_{ij}=1/m$. The processing time will be assumed to be a constant. In this case, a classical queueing model is made difficult due to the following reason. In conventional queueing models, the service center can start serving a new customer as soon as the last customer leaves. For core memories, due to the rewrite time, the memory has to wait. However, for tp≥tw, this problem can be surmounted by delaying the Pc in the Mp queue for an additional time equal to tw and reducing the effective execution time by the same amount. Hence, with this modification, the Mp can start serving the next Pc in its queue when the current Pc leaves the queue. But, with tp<tw, the Pc can make a new request before the memory that served it last recovers. Strictly speaking, a discrete Markov chain model can be formulated by making the basic time interval equal to the highest common factor of tp,tw, and ta. However, the number of possible epochs in a memory cycle and the size of the state vector and the number of states is very large.

4.1 AN APPROXIMATE MODEL FOR TP<TW

The results of the discrete Markov chain model of section 2.2 will be used to obtain an approximate of the execution rate. Consider an Mp cycle in which i Mp's are initially busy. If tp=tw, the i active Pc's make a request at the end of the cycle. However, if tp<tw, an active Pc makes a new request before the Mp that served it recovers. If this request is made to an Mp that is not busy, the new request can be served immediately. Consider an Mp cycle in which i Mp's are initially busy. During this cycle i Pc's are initially active. Now, these i Pc's can receive more service if they make their next request to the m-i idle Mp's. Now, if this request is made to an idle Mp, the effective service time (not including waiting time) for the active Pc is ta+tp. However, if this request is made to a busy Mp there is no increase in the execution rate due to the fact that tp<tw. Hence, the effective service time is tc.

Let b_i denote the probability that i Mp's are busy, obtained from the exact discrete Markov chain model for tp=tw. Now, the average number of idle Mp's that receive the next request is given by

$$(m-i) * ([1 - (1-1/m)^i)$$

Thus, the probability that an active Pc gets serviced by an idle Mp is

$$(m-i) * [1 - (1-1/m)^n] / i$$

Note that the above is a conditional probability based on the fact that i Pc's are active. Therefore, the unconditional probability that the effective service

TABLE 4.1

Average Number of Busy Mp's

tc=10 ta=tw=5

n × m	tp=5	tp=4	tp=3	tp=2	tp=1	tp=0
2 × 2	1.5000	1.5385	1.5790	1.6216	1.6667	1.7143
2 × 4	1.7500	1.8451	1.9512	2.0702	2.2047	2.3579
2 × 8	1.8750	2.0215	2.1928	2.3958	2.6403	2.9403
2 × 16	1.9375	2.1182	2.3362	2.6041	2.9414	3.3792
4 × 2	1.7500	1.7722	1.7949	1.8182	1.8421	1.8667
4 × 4	2.6210	2.6947	2.7832	2.8721	2.9669	3.0681
4 × 8	3.2652	3.4429	3.6410	3.8633	4.1145	4.4401
4 × 16	3.6268	3.9051	4.2297	4.6131	5.0729	5.6345
8 × 2	1.8750	1.8868	1.8987	1.9108	1.9231	1.9355
8 × 4	3.2657	3.3148	3.3654	3.4176	3.4714	3.5269
8 × 8	4.9471	5.1023	5.2676	5.4439	5.6324	5.8345
8 × 16	6.3149	6.6579	7.0403	7.4692	7.9539	8.5057
16 × 2	1.9375	1.9436	1.9497	1.9558	1.9620	1.9683
16 × 4	3.6270	3.6538	3.6811	3.7088	3.7369	3.7654
16 × 8	6.3154	6.4163	6.5204	6.6280	6.7392	6.8543
16 × 16	9.6258	9.9299	10.254	10.600	10.969	11.366

time of an active Pc is ta+tp is given by

$$\sum b_i * (m-i) * [1 - (1-1/m)^n] / i$$

Let f be the value of the above expression. The number of active Pc's is obtained from the discrete Markov chain model for tp=tw; denoted b / X. The expected value of the service time for an active Pc is

$$f * (ta+tp) + (1-f) * tc$$

Hence, the execution rate, expressed as unit instructions per second, is given by

$$X / [f*(ta+tp) + (1-f)*tc]$$

The average number of busy Mp cycles can be obtained by multiplying the above expression by tc.

Table 4.1 presents the results obtained for various values of m, n, ta, tp, and tc. Since this model is approximate it is meaningful to compare its results with a confidence interval obtained from simulation. Table 4.2 compares the results of this model and Strecker's model[StreW70] with simulation results. The results of this model are within 5% of the simulation. Figures 4.1 and 4.2 illustrate the effects of n and m on the MpAR.

4.2 STRECKER'S MODEL

Strecker[StreW70] uses his approximate model for tp=tw to analyze tp<tw. He

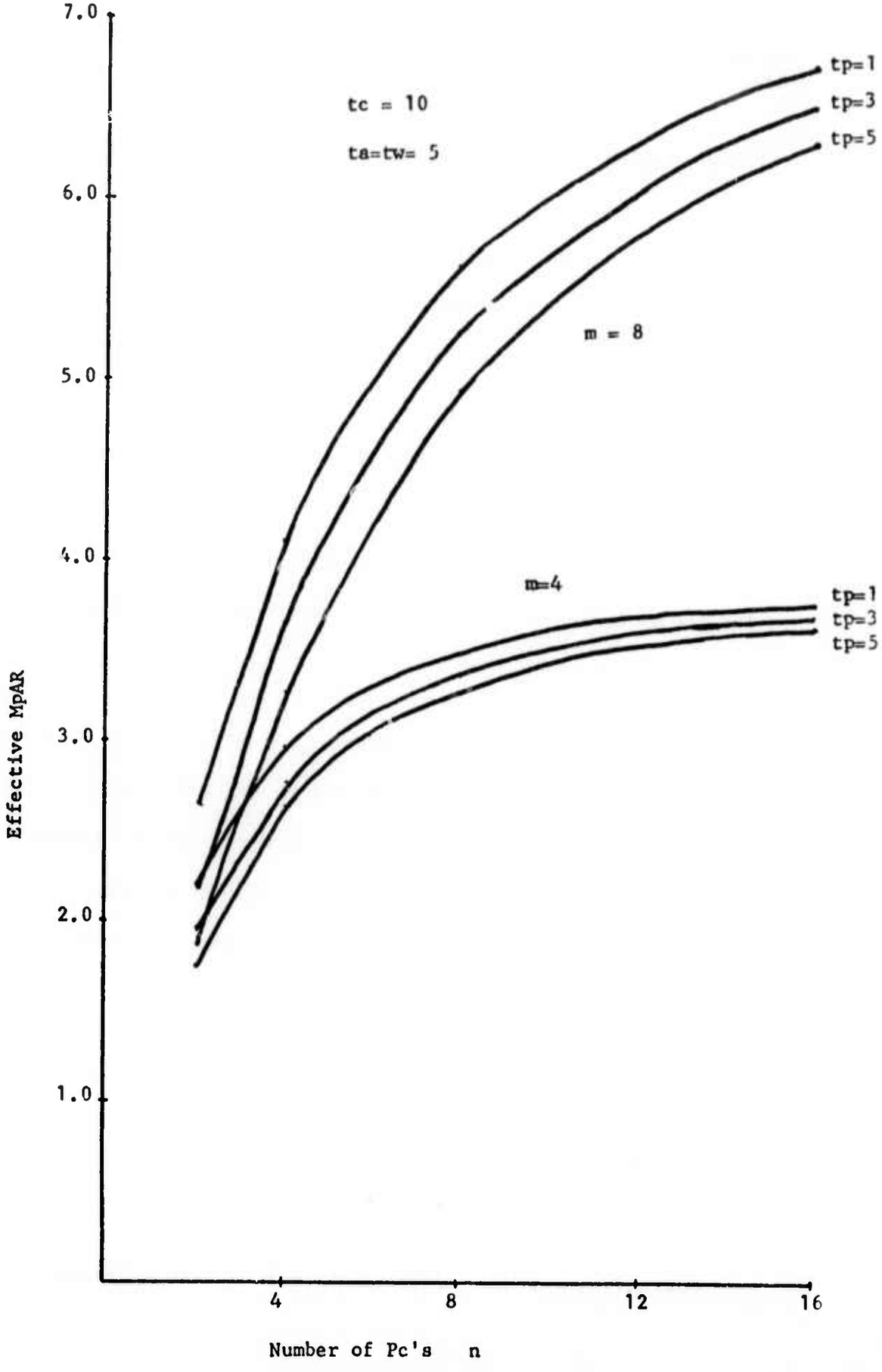


Figure 4.1 The effect of adding Pc's

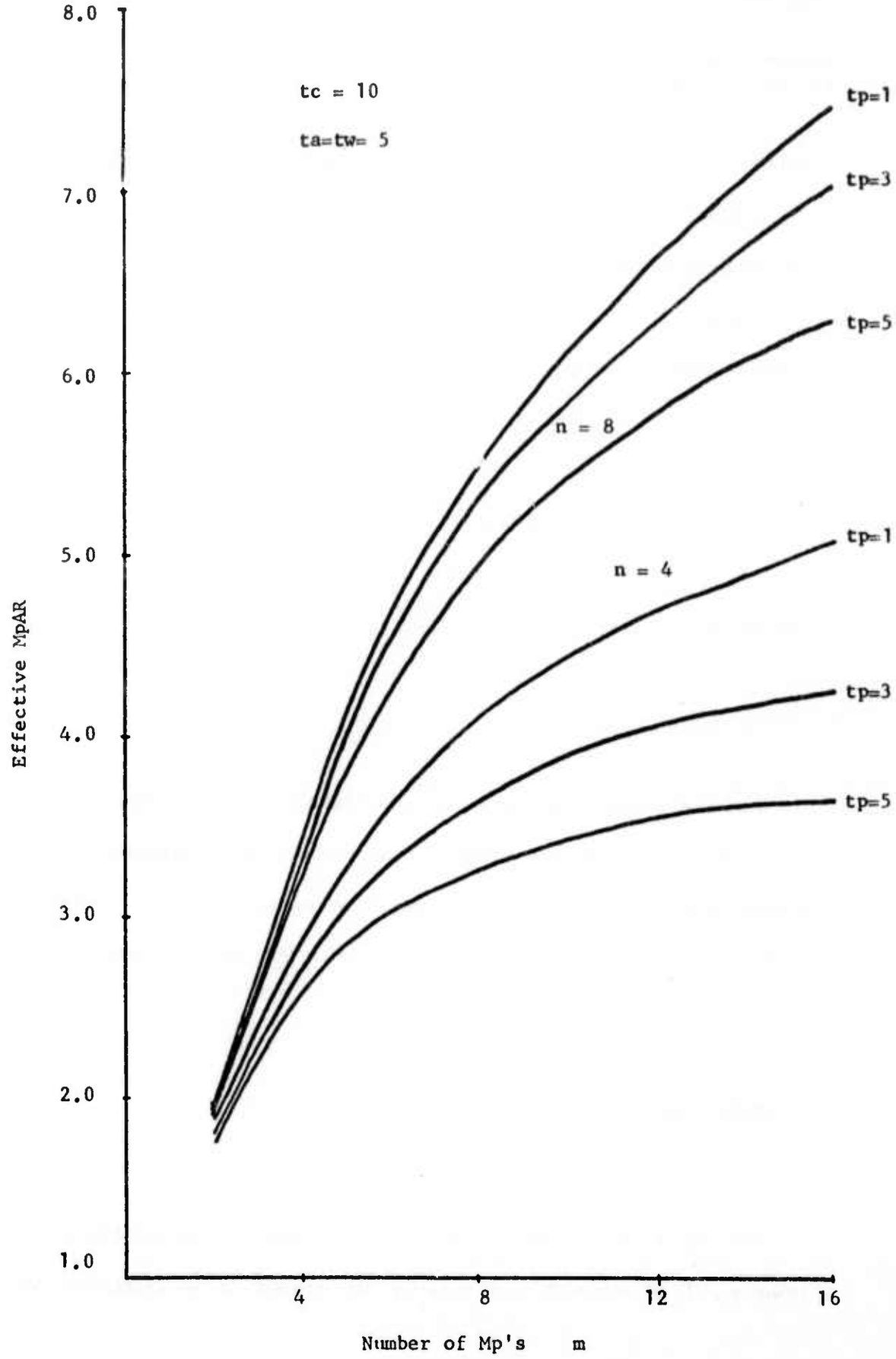


Figure 4.2 The effect of adding Mp's

defines the probability of an active Pc making a request to an occupied Mp as

p(occ) = average number of occupied Mp's / m

The probability of a request to an unoccupied Mp is

p(unocc) = 1 - p(occ)

Using his model for tp=tw,

$$p(occ) = 1 - (1-1/m)^n$$

Thus, the average amount of time required to execute an instruction is

$$E[t] = p(occ)*tc + p(unocc)*(ta+tp)$$

$$= tc + (1-1/m)^n * (tp-tw)$$

Therefore, the average number of busy Mp cycles is

$$tc * m*[1-(1-1/m)^n] / E[t]$$

Note that Strecker assumes a constant probability for an Mp being occupied. In the model of Sec. 4.1 the probabilty of an Mp being occupied depends on the number of busy Mp's during the cycle. Simulation results summarized in Table 4.2 show that the model proposed in section 4.1 is better than Strecker's model.

4.3 CONCLUDING REMARKS

With one Pc and one Mp there is no advantage gained by the fact that tp<tw. However, if an active Pc can make its next request to an unoccupied Mp

TABLE 4.2a

Average number of busy Mp's

Comparison of Analytic Models and Simulation Results

ta = tw = 5 tp = 1 tc = 10

n × m	Analytic Model (Section 4.1)	90% confidence interval from simulation	Strecker's Model
2 × 2	1.6667	(1.5591 , 1.6053)	1.6667
2 × 4	2.2047	(2.1342 , 2.2239)	2.2581
2 × 4	2.6403	(2.6039 , 2.6421)	2.7027
2 × 8	2.6403	(2.6039 , 2.6421)	2.7027
2 × 16	2.9414	(2.9240 , 2.9858)	2.9880
4 × 2	1.8421	(1.7746 , 1.7882)	1.9231
4 × 4	2.9669	(2.8102 , 2.9072)	3.1306
4 × 8	4.1145	(3.9982 , 4.1092)	4.3245
4 × 16	5.0729	(5.0587 , 5.1074)	5.2682
8 × 2	1.9231	(1.8514 , 1.9131)	1.9953
8 × 4	3.4714	(3.2850 , 3.3968)	3.7497
8 × 8	5.6324	(5.3417 , 5.5750)	6.0879
8 × 16	7.9539	(7.7336 , 7.9873)	8.4755
16 × 2	1.9620	(1.9225 , 1.9551)	2.0000
16 × 4	3.7369	(3.6047 , 3.7187)	3.9758
16 × 8	6.7392	(6.4601 , 6.5891)	7.4052
16 × 16	10.969	(10.5279,10.7559)	12.014

TABLE 4.2b

Average number of busy Mp's

Comparison of Analytic Models and Simulation Results

ta = tw = 5 tp = 3 tc = 10

n × m	Analytic Model (Section 4.1)	90% confidence interval from simulation	Strecker's Model
2 × 2	1.5790	(1.5368 , 1.5457)	1.5789
2 × 4	1.9512	(1.9058 , 2.0012)	1.9718
2 × 8	2.1928	(2.2019 , 2.2059)	2.2140
2 × 16	2.3362	(2.3441 , 2.3496)	2.3507
4 × 2	1.7949	(1.7626 , 1.7767)	1.8987
4 × 4	2.7832	(2.6950 , 2.8028)	2.9191
4 × 8	3.6410	(3.6021 , 3.6797)	3.7502
4 × 16	4.2297	(4.2319 , 4.2863)	4.3056
8 × 2	1.8987	(1.8530 , 1.9054)	1.9937
8 × 4	3.3654	(3.2646 , 3.3402)	3.6731
8 × 8	5.2676	(5.1470 , 5.2406)	5.6386
8 × 16	7.0403	(6.9457 , 7.1531)	7.3289
16 × 2	1.9497	(1.9212 , 1.9551)	2.0000
16 × 4	3.6811	(3.5241 , 3.7189)	3.9679
16 × 8	6.5204	(6.3928 , 6.4759)	7.2261
16 × 16	10.254	(9.9522 ,10.350)	11.093

significant increases in the execution rate can be obtained. The theoretical maximum execution rate for each Pc is 1/(ta+tp); the average number of busy Mp cycles is limited by n*tc/(ta+tp), if m is large enough; else it is bound by m.

CHAPTER 5

EMPIRICAL MEASUREMENTS, PARAMETER ESTIMATION AND MODEL VALIDATION

An important aspect in the development of mathematical models is the modeling process: the abstraction of the complex physical process to a model that is mathematically tractable. This chapter is devoted to validating the mathematical model.

One of the main parameters of the models developed in this thesis is the processing time. In order to evaluate the probability distribution and the mean of the processing time, measurements were made of the dynamic usage of the PDP-11/20 instruction set. B. Aygun's *D*ynamic *A*nalysis and *M*easurement *E*nvironment [AyguB73] was used to simulate the execution of over 34,500 PDP-11 instructions in carefully selected main-loop portions of four programs. The PDP-11 Processor Handbook, Interface Manual and Engineering Drawings were used to obtain the instruction timing for the various instructions and addressing modes. This information can be used to predict the performance of a multiprocessor system like C.mmp, which uses the PDP-11/20 as the processor. Note that the analytic models in this thesis are general; C.mmp is used in this chapter as a typical case for validation and illustration of the use of these models.

5.1 PDP-11/20 OVERVIEW

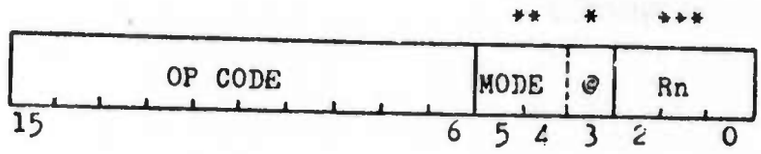
A brief discussion of the instruction timing and formats is presented here. The PDP-11/20 processor has five major states : ***fetch***, ***source***, ***destination***, ***execute*** and ***service***. The first four states are used during normal operation; service is used during special operations, such as traps and interrupts.

Fetch: locates and decodes an instruction. When fetch is completed, the processor enters another major state, depending on the type of instruction decoded. It is possible to go from fetch to any other state, including back to fetch. Every instruction starts by first entering the fetch state.

Source: decodes the source field of a double-operand instruction and transfers the source operand to the appropriate location. The source major state is entered only if the instruction is a double-operand type.

Destination: decodes the destination field of the appropriate instruction. Destination fields are present in both single and double-operand instructions. Destination operand is accessed and transferred to the appropriate location.

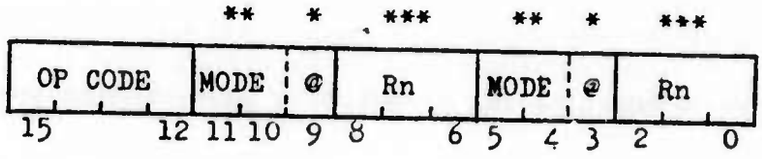
Execute: uses the data obtained during previous major states to perform the



Destination Address

* = Specifies Direct or Indirect Address
** = Specifies How Register will be used
*** = Specifies One of 8 General Purpose Registers

Figure 5.1a Single Operand Instruction Format



Source Address
Destination Address

* = Direct/Deferred Bit for Source and Destination Address
** = Specifies How Selected Registers are to be used
*** = Specifies a General Register

Figure 5.1b Double Operand Instruction Format

specified operation. During this state arithmetic operations, logic functions, and tests are performed, and the destination location is updated if required.

Service: used to execute special operations, such as interrupts, trap, etc.

Although the major states follow the sequence of fetch, source, destination, execute, and service, not all major states are required for every instruction. The processor enters only those major states necessary to execute the current instruction. The minimum sequence is from a fetch of one instruction directly to the fetch of the next instruction. The maximum sequence is fetch, source, destination, execute, service and back to fetch. The Interface Manual contains more detailed information about the states needed for various instructions.

The instruction format for all single operand instructions (such as clear, increment, test) is shown in Fig. 5.1a. Operations that imply two operands (such as add, subtract, move and compare) are handled by instructions that specify two addresses. The first operand is called the source operand, the second the destination operand. Bit assignments in the source and destination address fields may specify different modes and different general registers. The instruction format for the double operand instruction is depicted in fig 5.1b. Table 5.1a summarizes the four basic modes used with direct addressing; the four basic modes used with deferred addressing are described in Table 5.1b. The PDP-11 Processor Handbook contains numerous illustrative examples.

TABLE 5.1a

Direct Addressing Modes of PDP-11

Binary Code	Name	Assembler Syntax	Function
0 0 0	Register	Rn	Register contains operand.
0 1 0	Autoincrement	(Rn)+	Register is used as a pointer to sequential data then incremented.
1 0 0	Autodecrement	-(Rn)	Register is decremented then used as a pointer.
1 1 0	Index	X(Rn)	Value X is added to (Rn) to produce address of the operand. Neither X nor (Rn) is modified.

TABLE 5.1b

Deferred or Indirect Addressing Modes of PDP-11

Binary Code	Name	Assembler Syntax	Function
0 0 1	Register Deferred	@Rn or (Rn)	Register contains the address of the operand.
0 1 1	Autoincrement Deferred	@(Rn)+	Register is first used as a pointer to a word, then incremented by 2.
1 0 1	Autodecrement Deferred	@-(Rn)	Register is decremented by two and then used as a pointer to a word containing the address of the operand.
1 0 1	Index Deferred	@X(Rn)	Value X(stored in a word following the instruction) and (Rn) are added and the sum is used as a pointer to the word containing the address of the operand.

5.2 VALIDATION OF UNIT INSTRUCTION CONCEPT AND ESTIMATION OF TP

In this thesis, processor behavior has been modeled as an ordered sequence consisting of a memory request followed by some processing. A study of the PDP-11/20 instruction timing shown here exhibits a similar behavior.The access time starts when the Mp receives a request and ends when the data is received by the Pc.

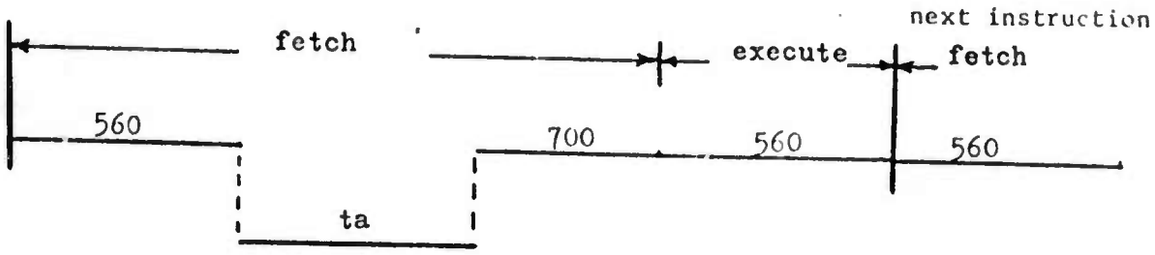
Each PDP-11/20 instruction can be broken down into sequences of unit instructions. Figure 5.2 shows the instruction timing for the various addressing modes and instruction types. The PDP-11 Processor Handbook lists the total time for executing the various instructions. The access time is assumed to be 450 ns. Figure 5.2 is consistent with the handbook. Table 5.2 summarizes the contribution of the different cases to the effective processing time. Note that these figures take into account the delay due to the PDP-11 Unibus. Table 5.3 contains the instruction mix obtained by using Aygun's *DAME*†. Table 5.4 gives the effective processing times and their relative frequencies; the cumulative probability distribution funct' n is plotted in Fig. 5.3. The access time of the memory was assumed to be 450ns and the basic clock period of the PDP-11/20 processor was taken to be its nominal value of 140ns. The average processing

---

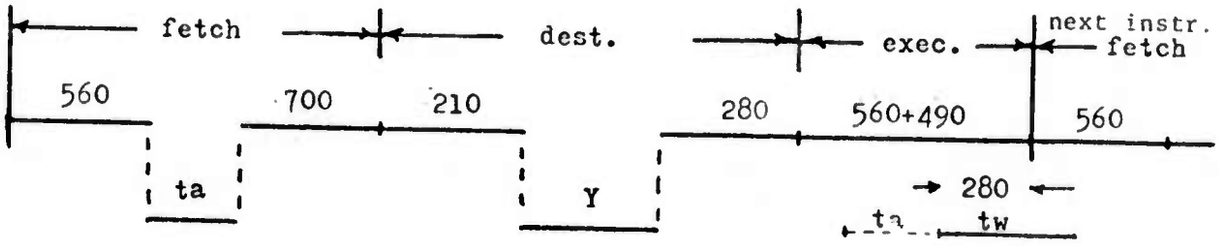
†The author acknowledges B. Aygun's assistance in obtaining the instruction mix.

Single Operand Instructions & Double Operand Instructions with srcmode=0

Dstmode=0



Dstmode≠0



Y :=

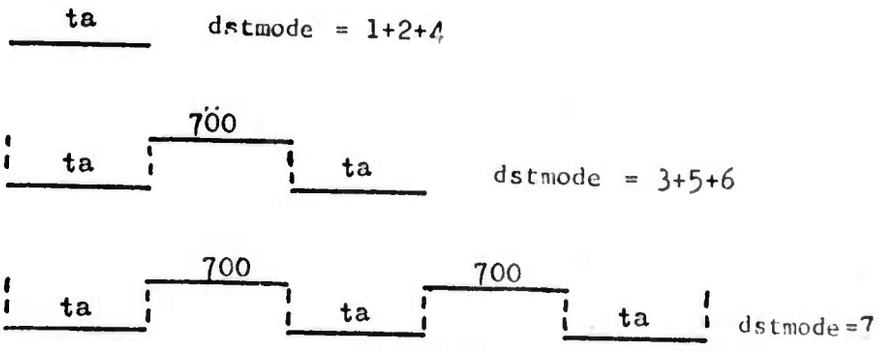
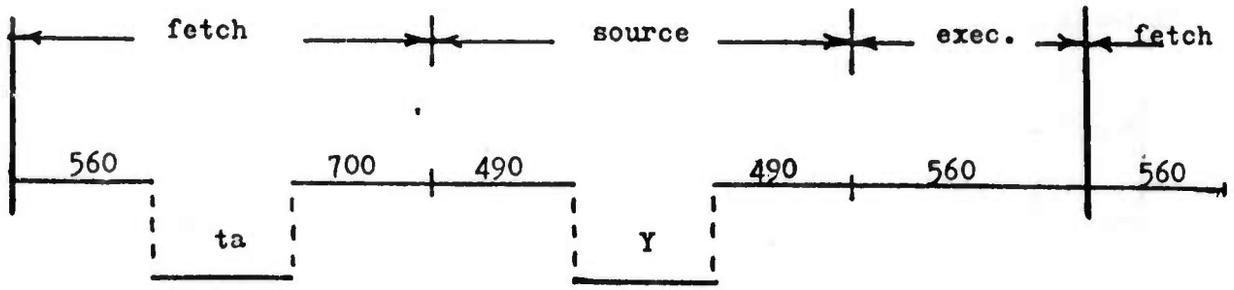


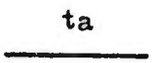
Figure 5.2 PDP-11/20 Instruction Timing

Double Operand Instructions with srcmode≠0

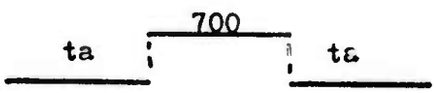
Dstmode=0 Srcmode≠0



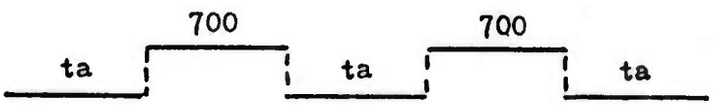
Y :=



srcmode = 1+2+4



srcmode = 3+5+6



srcmode=7

Dstmode≠0 Srcmode≠0

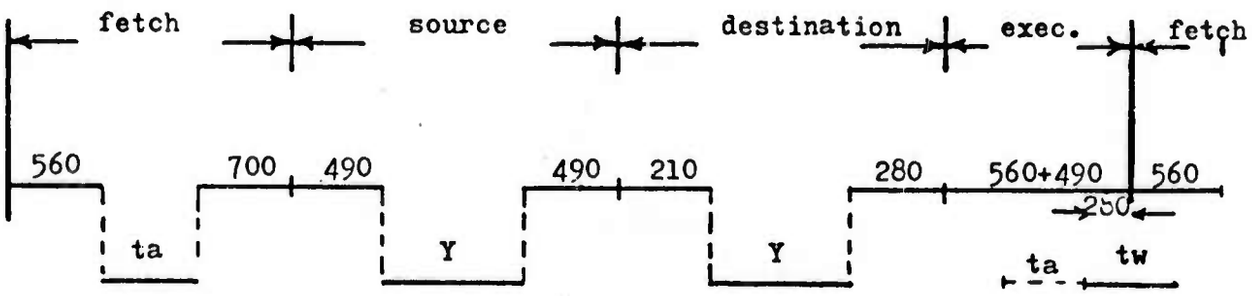
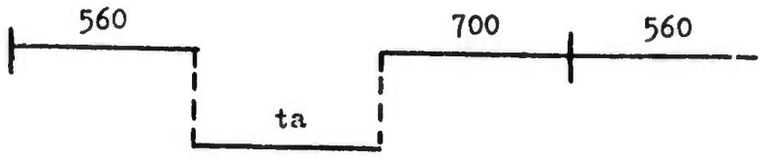


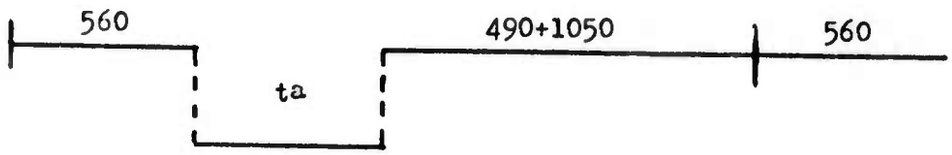
Fig. 5.2 (contd.)

Branch Instructions

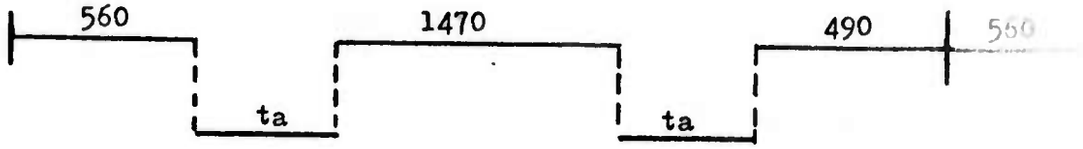
Unsuccessful



Sucessful



Return Subroutine



Jump to Subroutine

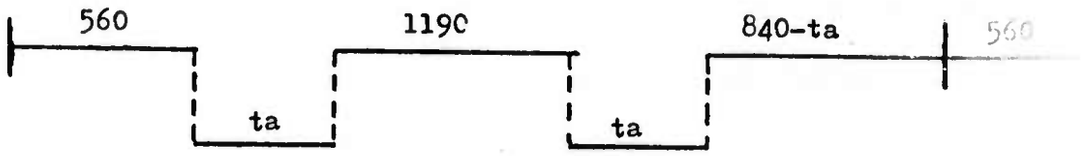


Fig 5.2 (contd.)

TABLE 5.2

Effective Processing Time

	tp in ns.	No. of occurrences
SINGLE OPERAND INSTRUCTIONS DOUBLE OPERAND WITH SRCMODE=0		
dstmode = 0		
	1820	1
dstmode = 1+2+4		
	910	1
	550	1
	840	1
dstmode = 3+5+6		
	910	1
	550	1
	840	1
	700	1
dstmode = 7		
	910	1
	550	1
	840	1
	700	2
DOUBLE OPERAND INSTRUCTIONS - SRCMODE≠0		
srcmode≠0 ∧ dstmode=0		
common to all cases		
	1190	1
	1610	1
srcmode = 3+5+6		
	700	1
srcmode = 7		
	700	2

srcmode≠0 ∧ dstmode≠0

common to all cases

1190	1
700	1
550	1
840	1

srcmode = 3+5+6

700	1

srcmode = 7

700	2

dstmode = 3+5+6

700	1

dstmode = 7

700	2

BRANCH INSTRUCTIONS

successful

2100	1

unsuccessful

1260	1

RETURN SUBROUTINE

1470	1
1050	1

JUMP SUBROUTINE

1190	1
970	1

TABLE 5.3

Instruction Mix

Single Operand Instructions	
Dstmode=0	19 %
Dstmode=1+2+4	10 %
Dstmode=3+5+6	3 %
Dstmode=7	.1%
Double Operand Instructions with srcmode=0	
Dstmode=0	4.2 %
Dstmode=1+2+4	5.75%
Dstmode=3+5+6	3.55%
Dstmode=7	0.0 %
Double Operand : srcmode≠0 . dstmode=0	
Srcmode=1+2+4	11.3%
Srcmode=3+5+6	3 %
Srcmode=7	0.1%
Double Operand : srcmode≠0 . dstmode≠0	6 %
Srcmode=3+5+6	2 %
Dstmode=3+5+6	2 %
Branch Instructions	
Successful	16 %
Unsuccessful	11 %
Return & Jump to Subroutine	
Jump	3 %
Return	3 %

TABLE 5.4

Relative Frequency Distribution of the Effective Processing Time

Value in ns.	Frequency
550	14.49 %
700	10.20 %
840	14.49 %
910	11.43 %
970	1.53 %
1050	1.53 %
1190	11.93 %
1260	5.61 %
1470	1.53 %
1610	7.35 %
1820	11.84 %
2100	8.16 %

Average Value = 1150 ns.

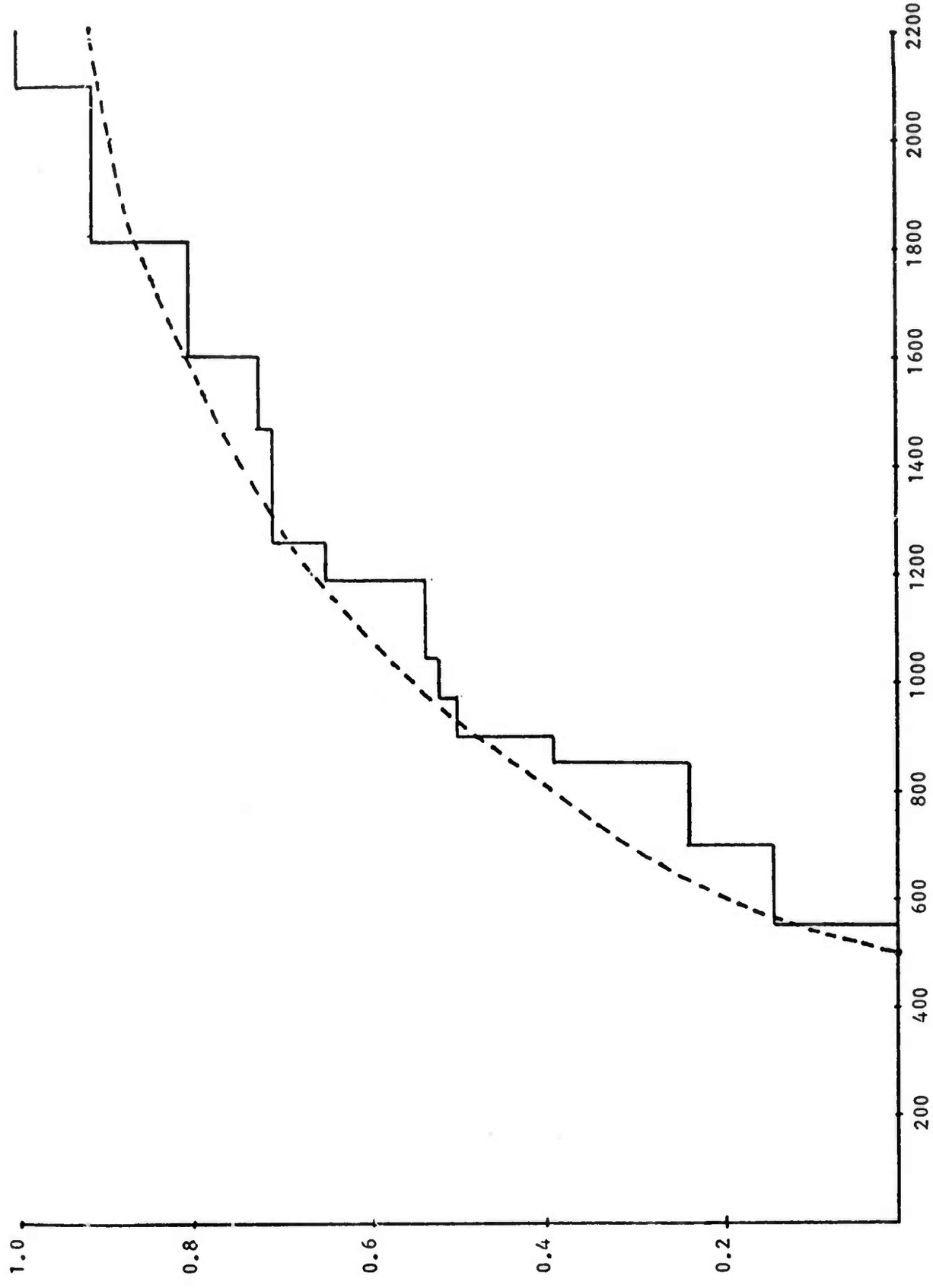


Figure 5.3 Cumulative probability distribution of the effective processing time of PDP-11/20

time obtained from this analysis is 1150 ns. The dotted curve in Fig. 5.3 is a shifted exponential distribution given by

$$\text{Prob}\{tp \le x\} = 1 - \exp[-(x-450)/700] \quad \text{for } x \ge 450$$
$$= 0 \quad \text{for } x < 450$$

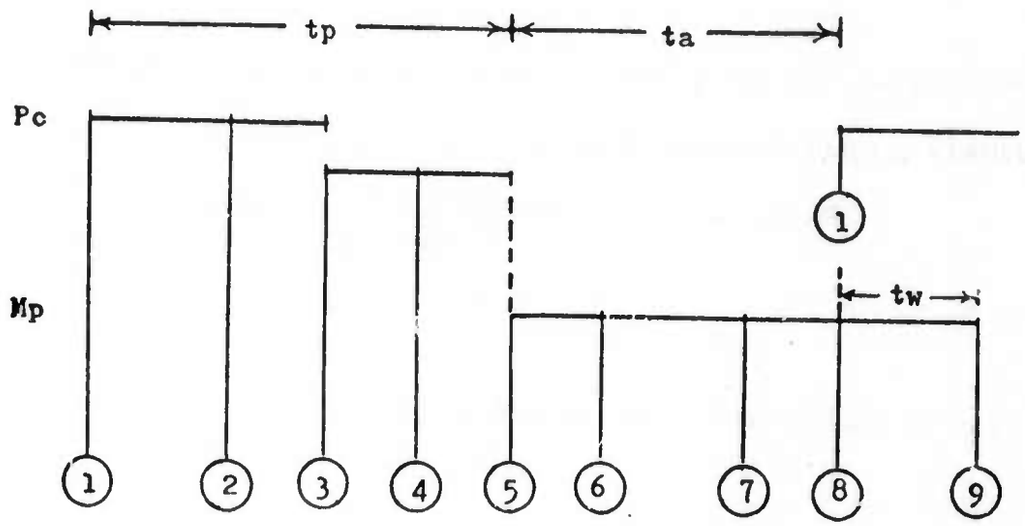
5.3 MODEL VALIDATION VIA C.mmp PERFORMANCE EVALUATION

In this section the models are used to predict the performance of C.mmp and compared with actual measurements. Figure 5.4 shows the effect of D.map and the crosspoint switch on the parameters tp, ta and tw. The access time of the C.mmp core memory is 250 ns. However, a 200 ns nominal switch and memory control delay yields an effective ta of 450 ns. Each Mp is 8-way interleaved. Thus the rewrite time of 400ns is overlapped with the next access if the next access is to one of the other 7 submodules of the Mp-module. Assuming random accessing within a module the average rewrite time experienced is only 50 ns. The effective average processing time is 1200 ns; the 50ns delay associated with the relocation registers D.map is added to the basic value of 1150 ns.

The following experiment†† was conducted on the partial realization of C.mmp available to date. A program was loaded into one Mp-module and the three

---

††C.Pierson and W.Broadley were instrumental in the experimental set-up.



Legend:

1	Pc receives data from Mp
2	Pc finishes operating on the data
3	Pc has address of new data
4	Dmap computes physical address of data and puts it on the Pc-Mp bus.
5	Memory has received request at the crosspoint switch
6	Memory controller (part of crosspoint switch) selects the request to be served and sets up switch.
7	Data read from storage location
8	Data sent through switch
9	Memory recovers and starts serving next request (if any)

Figure 5.4 Unit instruction timing diagram for C.mmp

available processors executed the code individually, in pairs of two, and collectively. The number of memory cycles was measured. Table 5.5 presents the results of the experiment. The analytic results predicted by the geometric model of section 5.2 are depicted in Table 5.6; simulation results with processing time having the shifted exponential distribution of Fig. 5.3 are also listed. The analytic results are very close to the measured performance. However, for the 3x1 multiprocessor case the analytic model is about 10% higher than the measured value. This is due to the read-modify-write cycles which make the Mp service time greater than tc. The effect of these read-modify-write cycles is not crucial when the interference is not excessive. The simulation and analytic results show that the performance can be predicted by simple mathematical models with reasonable accuracy.

TABLE 5.5

Summary of Measurements on C.mmp

Number of Mp accesses per second

(Millions/sec)

Pc[A]	Pc[B]	Pc[C]	Mp Access Rate
0	0	1	0.62015
0	1	0	0.61805
1	0	0	0.613657
0	1	1	1.14899
1	0	1	1.14672
1	1	0	1.14657
1	1	1	1.42466

0 - Pc is OFF

TABLE 5.6

Analytic Results

E[tp] = 1200 ns.

ta =450 ns. tc=500 ns.

Number of Pc's	Number of Mp's	Mp Access Rate (Millions/sec)	Simulation Results
1	1	0.60606	
2	1	1.14157	
3	1	1.56373	
4	4	2.39707	2.411
8	8	4.76009	4.751
16	16	9.48638	9.476

CHAPTER 6

CONCLUSIONS

In the previous chapters several analytic models have been presented. Chapter 2 was devoted to systems in which the effective processing time is equal to the memory rewrite time. An important result observed was the absence of a law of diminishing returns. The performance of a multiprocessor system with n processors and n memories continues to rise at a constant rate as n increases. A simple exponential server model showed this rate to be *0.5*; a constant processing time model predicted a slope of *0.586* for the average number of busy Mp's. The exponential server model gives the average number of busy Mp's as $n*m/(n+m-1)$. An approximate result for constant processing times gives the average number of busy Mp's as $i*[1-(1-1/i)^j]$, where i=max(n,m) and j=min(n,m). An intuitively obvious conclusion limits the maximum number of active Pc's by min(n,m). This maximum is reached if each processor accesses only one memory all the time. The model of section 2.6 analyzed the effect of skewing the processors' access patterns. Thus, the maximum $MpAR$ is $min(m,n)/tc$.

Chapter 3 contained several models for multiprocessor systems with tp>tw. A new model for geometrically distributed processing time was developed. A different analytic approach was used to model systems with private caches for the processors. In general, since the Pc is slow, it takes fewer memory units

for the performance to exhibit a saturation effect. This can be discerned by comparing Figs. 2.11 and 3.3a. In the absence of memory contention (which can now be possible even for m<n) the maximum $MpAR$ is $n/(ta+tp)$.

Chapter 4 discusses multiprocessor systems in which tp<tw. Since the processor is fast, performance improvement is obtained for m>n. If m=n, these systems do not yield significant improvement over systems with tp=tw. In general, adding an extra memory improves the performance more than adding an extra processor. The maximum average $MpAR$ is the minimum of m/tc and $n/(ta+tp)$; the maximum is achieved if the processors do not interfere. Note that since the Pc is very fast, it can make a request to the memory that served it last before the rewrite cycle is over. In this case, the Pc has to wait even though no other Pc is being serviced by the memory module.

Chapter 5 presented some empirical measurements of PDP-11 programs; C.mmp uses PDP-11's as the Pc's. It also illustrated how unit instructions can be extracted from the machine instructions. The measurements and the characterization of the processor's instruction timing was used to obtain the distribution of the effective instruction processing time, which was then used to evaluate analytically estimates of the unit execution rate of C.mmp. Contrary to common intuition, the unit instruction concept yields a larger effective processing time for register-register operations, since the processing time is a measure of the interval between two memory accesses. The average processing time was found to be 1150 ns. for the PDP-11/20. The average processing times for the

PDP-11/40 and 11/45 are approximately 625 and 400 nanoseconds. A PDP-11/20 instruction can require from 1 to 7 accesses to memory, but the average PDP-11/20 instruction makes approximately 2 accesses to memory; it comprises two unit instructions.

6.1 APPLICATIONS

The models developed in this thesis should give computer system designers considerable insight into some of the design issues. If tp≤tw, the performance (as indicated by the Mp/R) saturates for some value of m that is greater than n. If tp>tw, however, it may saturate in some cases for m<n. Most system designers tend to use optimality of cost/performance as an over-riding factor. Another important issue is the extendability or the effect of changing the system parameters[BhatS72]. Performance requirements often change and the initial design should be modifiable to meet the new performance desired. A good extendable design should therefore have a certain amount of unutilized capacity. The system should be designed to have a performance slightly greater than that which maximizes performance/cost.

In a multiprocessor system, the alternatives for change include increasing the number of Pc's or Mp's, as well as replacing the Mp's or Pc's with faster versions. The adding of Pc's and Mp's is a less viable alternative due to the

extent of the engineering effort involved. In some cases, the entire central crosspoint switch may have to be either redesigned or rebuilt. However, replacement with faster upward compatible processors or faster memories is easier. It is an undisputed fact that increasing the speed of the processor or memory by a factor of k will not enhance the performance by the same factor. The analytic models suggest that increasing the processor speed causes the saturation point to shift to a larger m. If $t_p \leq t_w$ a well-designed system is likely to have $m \geq n$. Since the memory is the limiting factor, the best choice for enhancing the performance is the use of faster memories. However, if the original design used more memory modules than needed to saturate the performance, faster processors will also yield some improvement. If $t_p \geq t_w$ substitution of faster processors effects an increase in the access rate. To allow for future substitution of faster processors it is desirable to choose $m \geq n$. Thus, a rule of thumb indicates that a good design choice is $m \geq n$, with a not too large a mismatch in Pc and Mp speeds. As described in chapter 5, C.mmp is a 16x16 multiprocessor system with t_p=1200 ns, t_a=450 ns and t_w=50 ns. The $MpAR$ expected is 9.476 million/sec. If the PDP-11/20 processor is replaced by a faster Pc (11/45) with a typical processing time of 450 ns the $MpAR$ increases to 15.6 million/sec. However, since the system is very much processor speed limited, a 150 ns cache for each Pc increases the $MpAR$ only by 14% and 20% for hit ratios of 0.5 and 0.7 respectively.

Chapter 6 : Conclusions
6.1 Applications

6.1.1 An Illustrative Example

In order to illustrate the use of the models developed in this thesis, consider a 8x8 multiprocessor system with tp=tw. This configuration is still memory limited, as seen from Fig. 2.10. Let the memory cycle time be 950 ns; access time=400 ns. Assume a switch delay of 200 ns and a 50 ns delay in the relocation hardware. The processor has a typical processing time of 450 ns. Hence, the effective processing time is 500 ns; effective access time is 600 ns; and the effective cycle time is 1100 ns (assuming that 50 ns of the switch delay is overlapped with the rewrite cycle).

The discrete Markov chain model of Chapter 2 gives the average number of busy Mp's to be 4.9471; the $MpAR$ is 4.4974 million/sec. Let us evaluate the effect of replacing the original Pc with either a 300 ns or a 150 ns processor. Note that the effective processing time is 350 ns and 200 ns. The corresponding percentage increase in the $MpAR$, as computed by the models of chapter 4, is 4.33% and 9.05%. This is not surprising since the original system was memory speed limited.

Instead of changing the Pc's, the Mp's could be changed. If the new Mp has ta=250 ns and tw=300 ns, the effective ta and tw are 450 ns and 250 ns respectively. The new cycle time is 700 ns. Since tp>tw the models of chapter 3 can be used. The new Mp access rate is 44.39% higher.

A third alternative is the use of a 150ns access time cache. The value of

α, the probability of finding the data in the cache, depends on the size of the cache. Consider three sizes that result in α=0.5, α=0.7 and α=0.8. The cache model of chapter 3 predicts performance enhancement of 67%, 107% and 130%.

The above performance data coupled with cost information can be used to select a profitable parameter change. When the number of design alternatives is large, simple analytic models help to determine a judicious choice.

6.2 PROPOSALS FOR FUTURE WORK

This thesis is not a panacea for multiprocessor system designers. An attempt has been made to develop some simple basic tools. Anyone working with systems is aware of their high degree of complexity and is likely to be shocked when he sees the simplicity of the models suggested. He may react negatively when he notices how much of the real system has been left out and how restrictive the assumptions are that have been made in the analysis. This skepticism is not entirely justified; simple analytic models often exhibit overall behavior similar to the complex system modeled.

One of the main assumptions made in this thesis is the independence of successive memory requests. In most real systems, due to program locality, there is some serial correlation between requests made by a Pc. Thus, if a Pc acresses

Mp[j] it continues to do so for some amount of time. A modeling technique suggested is dividing the system activities into various phases. The number of Mp's accessed in each phase is different. Jackson's general exponential server model[JackJ63] indicates that the stationary state probabilities depend only on the average frequency of visits to the various servers. Therefore the effect of the serial correlation of requests is not seen. However, for real systems which are not exponential, some degradation may be observed. We also assume $p_{ij}=1/m$. As indicated in section 2.6, this is not the most desirable access pattern. The effect of skewing the access patterns is to increase the *MpAR*. Given the above two opposing effects, $p_{ij}=1/m$ serves as a good parameter value for comparison at a high level.

Since no a priori empirical evidence is available, a large portion of future research activity should involve the measurement of real systems. This will bring to light the seriousness of the assumptions and establish a proper framework and area of applicability of analytic models. Measurement of dynamic program behavior should be stressed. It should be remembered that the output of the models can only be as good as the input. System analysts should not neglect the parameter estimation phase of performance prediction.

Future analytic studies should attempt to differentiate between instruction and data references. At a higher level, memory interference models can be used as a part of an overall hierarchical model of the computer system at the program level. Empirical studies should be conducted in order to obtain good concise

characterization of component and subsytem behavior. Such information can be used to drive more efficient simulations of complex systems.

REFERENCES

AndeJ62 Anderson, J.P. et al : D825 - A Multiple Computer System for Command and Control, *AFIPS* Proc. *FJCC*, Vol 22, pp.86-92, 1962.

AyguB73 Aygun, B. : Dynamic Analysis of Execution, Ph. D. thesis, Computer Science Dept., Carnegie-Mellon Univ., circa 1973.

BeckE64 Beckenbach, E. (editor): *Applied Combinatorial Mathematics*, Wiley, New York, 1964.

BellC71a Bell, C.G. and A.Newell : *Computer Structures: Readings and Examples*, Mcgraw-Hill, New York,1971.

BellC71b Bell, C.G. et al : C.mmp: The CMU Multiminiprocessor Computer, Dept. of Comp. Sci., Carnegie-Mellon Univ., Aug 1971.

BhatS72 Bhatia, S. : Techniques for Designing Balanced, Extensible Computer Systems, Ph.D. Thesis, Carnegie-Mellon Univ., Oct. 1972.

BuzeJ71 Buzen, J.B. : Queueing Network Models of Multiprogramming, Ph. D. Thesis, Harvard University, ESD-TR-71-345, August 1971.

FellW66 Feller, W. : *An Introduction to Probability Theory and its Applications*, Vol. 2, Wiley, New York , 1966.

GaveD67 Gaver, D.P. : Probability Models for Multiprogramming Computer Systems,*JACM*, Vol. 14, No. 3, July 1967, pp. 623-638.

GaveD71 Gaver, D.P. and G.S.Shedler : Control Variable Methods in the Simulation of a Multiprogrammed Computer System, *Naval Research Logistics Quarterly*, Vol. 18, No. 4. Dec. 1971, pp.435-450.

GordW67 Gordon, W.J. and G.F.Newell : Closed queueing systems with exponential servers, *Oper. Res.*, 15 (1967), pp 254-265.

GrenU72 Grenander, U. and R.F.Tsao : Quantitative Methods for Evaluating Computer System Performance: A Review and Proposal, *Statistical Computer Performance Evaluation* (ed. W.Freiberger), Academic Press, New York, 1972.

JackJ63 Jackson, J.R. : Jobshop-like queueing systems, *Management Sci.*, 10,1(Oct 1963), pp 131-142.

KobaH73 Kobayashi, H. : Application of the Diffusion Approximation to Queueing Networks: Part I - Equilibrium Queue Distributions, *1st Annual SIGME Conference* on Measurement and Evaluation, March 1973, pp 54-60.

LittJ61 Little, J. : A Proof of the Queueing Formula L=λW, *Operations Research*, Vol 9, No.3, pp. 383-387.

LiuC68 Liu, C.L. : *Introduction to Combinatorial Mathematics*, New York, McGraw-Hill, 1968.

McCrJ73 McCredie, J.W. : Analytic Models as Aids for Multiprocessor Design, Proc. of the *7th Annual Princeton Conference* on Information Science and Systems, March 1973.

MckiJ69 Mckinney, J.M. : A Survey of Analytic Time Sharing Models, *Computing Surveys*, Vol. 1, No. 2, pp.105-116, 1969.

MuntR72 Muntz, R.R. and F.Baskett : Queueing Network Models with Different Classes of customers, Proc. of *COMPCON 72*, Sept 72, pp 205-209.

NeweG71 Newell, G.F. : *Applications of Queueing Theory*, London, Chapman and Hall, 1971.

ParzE62 Parzen, E. : *Stochastic Processes*, Holden-Day, San Francisco,1962.

SkinC69 Skinner, C. and J.Asher : Effect of Storage Contention on System Performance, *IBM Sys. J.*, Vol 8,no. 4,1969, pp 319-333.

StreW70 Strecker, W.D. : Analysis of the Instruction Execution Rate in Certain Computer Structures, Ph.D. thesis, Carnegie-Mellon Univ., 1970.

WagnH69 Wagner, H. M. : *Principles of Operations Research*, Prentice-Hall, Englewood Cliffs, 1969.

WulfW72 Wulf, W.A. and C.G.Bell : C.mmp - A Multi-mini-processor, *AFIPS FJCC* Proc. 1972, Vol. 41, Part II, pp. 765-777.

APPENDIX A

```
C                                   APPENDIX A-1
C
C       LISTING OF PROGRAM FOR THE ALGORITHM SHOWN IN FIG. 2.5
C
C       ************************************************************************
C       DISCRETE MARKOV CHAIN TP=TW -- STATE(NO. OF STATES, NO. OF PC)
        INTEGER STACK(16,16),STATE(25,16),A(16),FIRST(16)
        1,IDONE(25),PTR(16),DONE(16),NWAYS(16),BOUND(17)
        DIMENSION TRANS(25,25),B(25),Z(25)
        COMMON TRANS,B,Z,NPARTS
        INTEGER SUM,T,X,Q
        TYPE 1973
1973                FORMAT(1X,'NUMBER OF PROCESSORS',/)
        ACCEPT 1974,NPC
1974                FORMAT(I)
        TYPE 1975
1975                FORMAT(1X,'NUMBER OF MEMORIES',/)
        ACCEPT 1974,NMP
        MINPM=MIN0(NPC,NMP)
C
        M=1
        PTR(NPC-1)=1
        GO TO 31
20      CONTINUE
        IF(NPC.GT.M)PTR(NPC-M)=NPARTS+1
        DO 21 I=2,M
21      A(I)=1
30      SUM=0
        DO 22 I=2,M
22      SUM=SUM+A(I)
31      A(1)=NPC-SUM
C
C       WRITE(15,2000),(A(I),I=1,M)
2000                FORMAT(1X,16(I3,1X))
        NPARTS=NPARTS+1
        DO 17 K=1,M
17      STATE(NPARTS,K)=A(K)
C
        T=2
60      CONTINUE
        IF(T.GT.M)GO TO 120
        X=A(1)-A(T)
        IF(X.GT.1)GO TO 100
        T=T+1
```

Appendix A-1
Listing of Program for the Algorithm shown in Fig. 2.5

```
        GO TO 60
100                 CONTINUE
        ITMP=A(T)
        DO 101 I=2,T
101                 A(I)=ITMP+1
        GO TO 30
120                 M=M+1
        BOUND(M)=NPARTS
C       WRITE(15,2001),NPARTS
2001                FORMAT(/,1X,'********** ',I3,'**********',/)
        IF(M.LE.MINPM)GO TO 20
        WRITE(15,111),NPARTS
111                 FORMAT(/,1X,'NUMBER OF PARTITIONS=',I)
        WRITE(15,3113)
3113                FORMAT(///)
C
C
C
C       INITIALIZE THE STACK, NWAYS, FIRST
C
        DO 2100 I=1,NPC-1
        STACK(I,1)=I
        NWAYS(I)=1
2100                FIRST(I)=2
        FIRST(NPC-1)=1
C
C       BEGIN WALK THROUGH TREE
C
        L=NPC
1       CONTINUE
        IF(DONE(L).EQ.1)GO TO 25
C
200                 K=L-1
        J=FIRST(K)
C
        DO 10 I=J,NMP
        IF(STACK(K,I).NE.STACK(K,J))GO TO 15
10      CONTINUE
C
        DONE(L)=1
        FIRST(K)=1
        I=I+1
C
15      NWAYS(L)=I-J
C
        DO 300 KK=1,NMP
```

```
300                     STACK(L,KK)=STACK(K,KK)
        STACK(L,J)=STACK(K,J)+1
C
        IF(DONE(L).NE.1)FIRST(K)=I
C
        IF(L.EQ.NPC)GO TO 1000
C
C
C       SET PTR TO ORIGINAL STATE AT LEVEL L
C
        IF(PTR(L).NE.0)IDONE(PTR(L))=1
        K=NPC-L
        K1=BOUND(K)+1
        K2=BOUND(K+1)
C
        DO 35 JJ=K,1,-1
C
        DO 32 KK=K1,K2
        IF(STATE(KK,JJ).EQ.STACK(L,JJ)+1)GO TO 34
32      CONTINUE
C
        KK=0
        GO TO 36
34      K1=KK
C
35      CONTINUE
C
36      PTR(L)=KK
C
C
C
        L=L+1
        GO TO 200
C
C
25      DONE(L)=0
        L=L-1
        IF(L.EQ.1)GO TO 7777
C
        GO TO 1
C
C
C       FIND TERMINAL STATE
C
1000                CONTINUE
C
```

Appendix A-1
Listing of Program for the Algorithm shown in Fig. 2.5

```
      DO 1100 K=1,NMP
      IF(STACK(NPC,K).EQ.0)GO TO 1150
1100              CONTINUE
      K=NMP+1
C
1150              K=K-1
      K1=BOUND(K)+1
      K2=BOUND(K+1)
C
      DO 135 JJ=K,1,-1
C
      DO 132 KK=K1,K2
      IF(STATE(KK,JJ).EQ.STACK(NPC,JJ))GO TO 134
132               CONTINUE
C
      TYPE 9876
9876              FORMAT(1X,'CANNOT FIND TERMINAL STATE !!!!!!')
C
134               K1=KK
C
135               CONTINUE
C
C     UPDATE TRANSITION MATRIX
C
      TEMP=1
      DO 150 I=NPC-1,1,-1
      II=PTR(I)
      TEMP=TEMP*NWAYS(I+1)
      IF(II.EQ.0)GO TO 150
      IF(IDONE(II).NE.1)TRANS(KK,II)=TRANS(KK,II)+TEMP
150               CONTINUE
C
      IF(NPC.LE.NMP)TRANS(KK,NPARTS)=TRANS(KK,NPARTS)+TEMP*NMP
C
      GO TO 1
C
C
C     TRANSITION MATRIX HAS BEEN GENERATED
C
7777              CONTINUE
C     DO 5544 I=1,NPARTS
C5544             WRITE(15,4455),(TRANS(I,J),J=1,NPARTS)
C4455             FORMAT(1X,25(F8.4,1X))
C
      DO 7250 I=1,MINPM
      Y=NMP***I
```

```
      DO 7250 J=BOUND(I)+1,BOUND(I+1)
      DO 7100 K=1,NPARTS
7100              TRANS(K,J)=TRANS(K,J)/Y
7250              CONTINUE
C     DO 5544 I=1,NPARTS
C5544             WRITE(15,4455),(TRANS(I,J),J=1,NPARTS)
4455              FORMAT(1X,25(F8.6,1X))
C
      DO 8000 I=1,NPARTS
      TRANS(I,I)=TRANS(I,I)-1.0
8000              TRANS(NPARTS,I)=1.0
C
      B(NPARTS)=1
C
      CALL GAUSS
C
      DO 8500 I=1,MINPM
C
      TMP=0
      DO 8250 J=BOUND(I)+1,BOUND(I+1)
8250              TMP=TMP+Z(J)
C
8500              OCC=OCC+TMP*I
C
      WRITE(15,8750),OCC
8750              FORMAT(//,1X,'EXPECTED VALUE OF NO. OF BUSY MP= '
      1,F10.6)
C
      STOP
      END
C
C
      SUBROUTINE GAUSS
C     THIS SUBROUTINE SOLVES SIMULTANEOUS LINEAR EQUATIONS
C                 A*X=B
C
      DIMENSION A(25,25),B(25),X(25)
      COMMON A,B,X,N
      INTEGER S
C
      S=N
5     CONTINUE
      IF(S-1) 50,50,10
10    CONTINUE
      IF(S.GT.2)GO TO 105
      D=A(1,1)*A(2,2)-A(1,2)*A(2,1)
```

Appendix A-1
Listing of Program for the Algorithm shown in Fig. 2.5

```
      IF(ABS(D).GT.0.0005)GO TO 105
      TYPE 25
      GO TO 100
105                DO 20 I=1,S
      M=S-I+1
      IF(ABS(A(M,S)).GT.0.0005)GO TO 30
20    CONTINUE
      TYPE 25
25    FORMAT(1X,'THE COEFFICIENT MATRIX IS SINGULAR',/)
      GO TO 100
30    CONTINUE
      IF(M.EQ.S)GO TO 40
      T=B(S)
      B(S)=B(M)
      B(M)=T
      DO 35 J=1,S
      T=A(S,J)
      A(S,J)=A(M,J)
      A(M,J)=T
35    CONTINUE
40    CONTINUE
      DO 45 I=1,S-1
      K=S-I
      IF(ABS(A(S,S)).GT.0.0005)GO TO 42
      GO TO 100
42    B(K)=B(K)-B(S)*A(K,S)/A(S,S)
      DO 45 J=1,S-1
      A(K,J)=A(K,J)-A(K,S)*A(S,J)/A(S,S)
45    CONTINUE
      S=S-1
      GO TO 5
50    CONTINUE
      DO 70 I=1,N
      SUM=B(I)
      DO 60 J=1,I-1
      SUM=SUM-A(I,J)*X(J)
60    CONTINUE
      IF(ABS(A(I,I)).GT.0.0005)GO TO 61
      GO TO 100
61    X(I)=SUM/A(I,I)
70    CONTINUE
100                CONTINUE
      RETURN
      END
```

```
C                                   APPENDIX A-2
C
C       LISTING OF PROGRAM FOR THE ALGORITHM SHOWN IN FIG. 2.8
C
C       ***********************************************************************
C
C       THIS PROGRAM SOLVES THE DISCRETE MARKOV CHAIN FOR TP=TW
C       USING SINGLE LEVEL TRANSITIONS
C       LONG VERSION
C
C       ***********************************************************************
        INTEGER STATE(186,16),BOUND(17),B1(17),B2(17)
        INTEGER S1(116,16),S2(146,16)
        DIMENSION TRANS(186,186),B(186),Z(186),X12(146,116),X23(186,146)
        DIMENSION C(186)
        COMMON NPARTS,TRANS,B,Z
        INTEGER T,Q
        INTEGER A(16),BD(17)
        TYPE 1973
1973                FORMAT(1X,'NUMBER OF PROCESSORS',/)
        ACCEPT 1974,NPC
1974                FORMAT(I)
        TYPE 1975
1975                FORMAT(1X,'NUMBER OF MEMORIES',/)
        ACCEPT 1974,NMP
        WRITE(15,1949),NPC,NMP
1949                FORMAT(1X,'NUMBER OF PROCESSORS =',I2,/,
        11X,'NUMBER OF MEMORIES =',I2,/)
        MINPM=MIN0(NPC,NMP)
        LEVEL=NPC
C
        IPRT=1
        GO TO 4000
C
C       PARTITION NPC INTO NMP PARTS
C       FINAL STATE VECTOR --- STATE
C
421                 NPARTS=NNPART
        DO 4421 I=1,M
4421                BOUND(I)=BD(I)
C
        IPRT=2
        LEVEL=NPC-1
        GO TO 4000
C
C       PARTITION NPC-1
```

Appendix A-2
Listing of Program for the Algorithm shown in Fig. 2.8

```
C       PARTIAL STATE VECTOR --- S2
C
422                 N2=NNPART
        DO 4422 I=1,M
4422                B2(I)=BD(I)
C
C       GENERATE NUMBER OF WAYS OF GOING FROM S2 TO STATE
C
        LSRC=MIN0(NPC-1,NMP)
        DO 1 I=1,NPARTS
        DO 1 J=1,N2
1       X23(I,J)=0
        DO 40 K=1,LSRC
        I1=B2(K)+1
        I2=B2(K+1)
        DO 40 L=I1,I2
        K1=BOUND(K)+1
        K2=BOUND(K+2)
        IF(MINPM.LT.K+1)K2=BOUND(K+1)
        DO 32 KK=K1,K2
        M=0
        JJ=MIN0(MINPM,K+1)
        DO 20 I=1,JJ
        IF(S2(L,I).EQ.STATE(KK,I))GO TO 20
        IF(S2(L,I).NE.STATE(KK,I)-1)GO TO 32
        M=M+1
        II=I
20      CONTINUE
        IF(M.NE.1)GO TO 32
        DO 25 I=II,NMP
        IF(S2(L,I).NE.S2(L,II))GO TO 27
25      CONTINUE
        I=I+1
27      X23(KK,L)=I-II
32      CONTINUE
40      CONTINUE
C
C       UPDATE TRANSITION MATRIX --- TRANS
C
        K=1
        K1=BOUND(K)+1
        K2=BOUND(K+1)
C
        DO 100 L=1,N2
C
        DO 35 JJ=K,1,-1
```

```
C
      DO 5632 KK=K1,K2
      IF(STATE(KK,JJ).EQ.S2(L,JJ)+1)GO TO 34
5632              CONTINUE
C
      KK=0
      GO TO 36
34    K1=KK
C
35    CONTINUE
C
36    CONTINUE
      IF(KK.EQ.0)GO TO 100
      DO 50 I=1,NPARTS
50    TRANS(I,KK)=TRANS(I,KK)+X23(I,L)
100               CONTINUE
C
C     DECREMENT LEVEL
C
      LEVEL=NPC-2
200               CONTINUE
      IF(LEVEL.EQ.0)GO TO 300
C
C     PARTITION LEVEL
C     PARTIAL STATE VECTOR --- S1
C
      IPRT=3
      GO TO 4000
423               N1=NNPART
      DO 4423 I=1,M
4423              B1(I)=BD(I)
C
C     COMPUTE NWAYS FOR GOING FROM S1 TO S2
C
      LSRC=MIN0(LEVEL,NMP)
C
      DO 341 I=1,N2
      DO 341 J=1,N1
341               X12(I,J)=0
      DO 5740 K=1,LSRC
      I1=B1(K)+1
      I2=B1(K+1)
      DO 5740 L=I1,I2
      K1=B2(K)+1
      K2=B2(K+2)
      IF(MINPM.LT.K+1)K2=B2(K+1)
```

Appendix A-2

Listing of Program for the Algorithm shown in Fig. 2.8

```
      DO 5732 KK=K1,K2
      M=0
      JJ=MIN0(MINPM,K+1)
      DO 577 I=1,JJ
      IF(S1(L,I).EQ.S2(KK,I))GO TO 577
      IF(S1(L,I).NE.S2(KK,I)-1)GO TO 5732
      M=M+1
      II=I
577                 CONTINUE
      IF(M.NE.1)GO TO 5732
      DO 5725 I=II,NMP
      IF(S1(L,I).NE.S1(L,II))GO TO 979
5725                CONTINUE
      I=I+1
979                 X12(KK,L)=I-II
5732                CONTINUE
5740                CONTINUE
C
C     MATRIX MULTIPLICATION TO GET NWAYS FROM S1 TO STATE
C
      IC=1
      GO TO 125
1255                CONTINUE
C
C     UPDATE TRANS
C
      K=NPC-LEVEL
      IF(K.GT.MINPM)GO TO 5800
      K1=BOUND(K)+1
      K2=BOUND(K+1)
C
      DO 5800 L=1,N1
C
      DO 5835 JJ=K,1,-1
C
      DO 5832 KK=K1,K2
      IF(STATE(KK,JJ).EQ.S1(L,JJ)+1)GO TO 5834
5832                CONTINUE
C
      KK=0
      GO TO 5836
5834                K1=KK
C
5835                CONTINUE
C
5836                CONTINUE
```

```
      IF(KK.EQ.0)GO TO 5800
      DO 5850 I=1,NPARTS
5850                TRANS(I,KK)=TRANS(I,KK)+X23(I,L)
5800                CONTINUE
C
C     DECREMENT LEVEL
C
      LEVEL=LEVEL-1
      IF(LEVEL.EQ.0)GO TO 300
C
C     PARTITION LEVEL
C     PARTIAL STATE VECTOR --- S2
C
      IPRT=4
      GO TO 4000
424                 N2=NNPART
      DO 4424 I=1,M
4424                B2(I)=BD(I)
C
C     COMPUTE NWAYS FROM S2 TO S1
C
      LSRC=MIN0(LEVEL,NMP)
C
      DO 591 I=1,N1
      DO 591 J=1,N2
591                 X12(I,J)=0
      DO 5940 K=1,LSRC
      I1=B2(K)+1
      I2=B2(K+1)
      DO 5940 L=I1,I2
      K1=B1(K)+1
      K2=B1(K+2)
      IF(MINPM.LT.K+1)K2=B1(K+1)
      DO 5932 KK=K1,K2
      M=0
      JJ=MIN0(MINPM,K+1)
      DO 5920 I=1,JJ
      IF(S2(L,I).EQ.S1(KK,I))GO TO 5920
      IF(S2(L,I).NE.S1(KK,I)-1)GO TO 5932
      M=M+1
      II=I
5920                CONTINUE
      IF(M.NE.1)GO TO 5932
      DO 5925 I=II,NMP
      IF(S2(L,I).NE.S2(L,II))GO TO 5927
5925                CONTINUE
```

Appendix A-2
Listing of Program for the Algorithm shown in Fig. 2.8

```
          I=I+1
5927                X12(KK,L)=I-II
5932                CONTINUE
5940                CONTINUE
C
C       MATRIX MULTIPLICATION GIVES NWAYS FROM S2 TO STATE
C
          IC=2
          GO TO 125
1260                CONTINUE
C
C       UPDATE TRANS
C
          K=NPC-LEVEL
          IF(K.GT.MINPM)GO TO 6300
          K1=BOUND(K)+1
          K2=BOUND(K+1)
C
          DO 6300 L=1,N2
C
          DO 6335 JJ=K,1,-1
C
          DO 6332 KK=K1,K2
          IF(STATE(KK,JJ).EQ.S2(L,JJ)+1)GO TO 6334
6332                CONTINUE
C
          KK=0
          GO TO 6336
6334                K1=KK
C
6335                CONTINUE
C
6336                CONTINUE
          IF(KK.EQ.0)GO TO 6300
          DO 6350 I=1,NPARTS
6350                TRANS(I,KK)=TRANS(I,KK)+X23(I,L)
6300                CONTINUE
C
C       DECREMENT LEVEL
C
          LEVEL=LEVEL-1
          GO TO 200
300                 CONTINUE
          IF(NPC.GT.NMP)GO TO 400
          DO 350 I=1,NPARTS
350                 TRANS(I,NPARTS)=TRANS(I,NPARTS)+1.0*NMP*X23(I,1)
```

```
400                 CONTINUE
C
C
C       TRANSITION MATRIX HAS BEEN GENERATED
C
7777                CONTINUE
C       DO 5544 I=1,NPARTS
C5544               WRITE(15,4455),(TRANS(I,J),J=1,NPARTS)
C4455               FORMAT(1X,25(F8.4,1X))
C
        DO 7250 I=1,MINPM
        Y=(1.0*NMP)**I
        DO 7250 J=BOUND(I)+1,BOUND(I+1)
        DO 7100 K=1,NPARTS
7100                TRANS(K,J)=TRANS(K,J)/Y
7250                CONTINUE
C       DO 5544 I=1,NPARTS
C5544               WRITE(15,4455),(TRANS(I,J),J=1,NPARTS)
C4455               FORMAT(1X,25(F8.6,1X))
C
        DO 8000 I=1,NPARTS
        TRANS(I,I)=TRANS(I,I)-1.0
8000                TRANS(NPARTS,I)=1.0
C
        B(NPARTS)=1
C
        CALL GAUSS
C
        DO 8500 I=1,MINPM
C
        TMP=0
        DO 8250 J=BOUND(I)+1,BOUND(I+1)
8250                TMP=TMP+Z(J)
        WRITE(15,9321),I,TMP
9321                FORMAT(1X,'PROB(NO. OF BUSY MEMS=',I2,')=',1X,F10.6)
C
8500                OCC=OCC+TMP*I
C
        WRITE(15,8750),OCC
8750                FORMAT(//,1X,'EXPECTED VALUE OF NO. OF BUSY MP= '
        1,F10.6)
C
        STOP
C
C       MATRIX MULTIPLICATION IN-LINE SUBROUTINE
C
```

Appendix A-2
Listing of Program for the Algorithm shown in Fig. 2.8

```
125                 CONTINUE
        N=NPARTS
        NM=N2
        M=N1
        IF(IC.EQ.1)GO TO 126
        NM=N1
        M=N2
126                 CONTINUE
C
        DO 511 I=1,N
        DO 5111 J=1,M
        C(J)=0
        DO 5111 K=1,NM
5111                C(J)=C(J)+X23(I,K)*X12(K,J)
        DO 511 J=1,M
511                 X23(I,J)=C(J)
        IF(IC.EQ.1)GO TO 1255
        GO TO 1260
        STOP
C
C
C       IN-LINE SUBROUTINE FOR GENERATING PARTITIONS
C                   STATE VECTORS
C
C
4000                MINL=MIN0(LEVEL,NMP)
C
        NNPART=0
        SUM=0
        M=1
        GO TO 31
220                 CONTINUE
        DO 21 I=2,M
21      A(I)=1
30      SUM=0
        DO 22 I=2,M
22      SUM=SUM+A(I)
31      A(1)=LEVEL-SUM
C
C       WRITE(15,2000),(A(I),I=1,M)
2000                FORMAT(1X,16(I3,1X))
        NNPART=NNPART+1
        GO TO (171,172,173,172),IPRT
171                 CONTINUE
        DO 1711 K=1,M
1711                STATE(NNPART,K)=A(K)
```

```
      IF(NMP.LE.M)GO TO 1700
      DO 181 K=M+1,NMP
181                 STATE(NNPART,K)=0
      GO TO 1700
172                 CONTINUE
      DO 1722 K=1,M
1722                S2(NNPART,K)=A(K)
      IF(NMP.LE.M)GO TO 1700
      DO 182 K=M+1,NMP
182                 S2(NNPART,K)=0
      GO TO 1700
173                 CONTINUE
      DO 1733 K=1,M
1733                S1(NNPART,K)=A(K)
      IF(NMP.LE.M)GO TO 1700
      DO 183 K=M+1,NMP
183                 S1(NNPART,K)=0
1700                T=2
60    CONTINUE
      IF(T.GT.M)GO TO 1220
      IIX=A(1)-A(T)
      IF(IIX.GT.1)GO TO 1000
      T=T+1
      GO TO 60
1000                CONTINUE
      ITMP=A(T)
      DO 101 I=2,T
101                 A(I)=ITMP+1
      GO TO 30
1220                M=M+1
      BD(M)=NNPART
C     WRITE(15,2201),NNPART
2201                FORMAT(/,1X,'********** ',I3,'**********',/)
      IF(M.LE.MINL)GO TO 220
      GO TO (421,422,423,424),IPRT
C
C
      END
C
      SUBROUTINE GAUSS
C     THIS SUBROUTINE SOLVES SIMULTANEOUS LINEAR EQUATIONS
C                   A*X=B
C
      DIMENSION A(186,186),B(186),X(186)
      COMMON N,A,B,X
      INTEGER S
```

Appendix A-2
Listing of Program for the Algorithm shown in Fig. 2.8

```
C
      S=N
5     CONTINUE
      IF(S-1) 50,50,10
10    CONTINUE
      IF(S.GT.2)GO TO 105
      D=A(1,1)*A(2,2)-A(1,2)*A(2,1)
      IF(ABS(D).GT.0.0005)GO TO 105
      TYPE 25
      GO TO 100
105                DO 20 I=1,S
      M=S-I+1
      IF(ABS(A(M,S)).GT.0.0005)GO TO 30
20    CONTINUE
      TYPE 25
25    FORMAT(1X,'THE COEFFICIENT MATRIX IS SINGULAR',/)
      GO TO 100
30    CONTINUE
      IF(M.EQ.S)GO TO 40
      T=B(S)
      B(S)=B(M)
      B(M)=T
      DO 35 J=1,S
      T=A(S,J)
      A(S,J)=A(M,J)
      A(M,J)=T
35    CONTINUE
40    CONTINUE
      DO 45 I=1,S-1
      K=S-I
      IF(ABS(A(S,S)).GT.0.0005)GO TO 42
      GO TO 100
42    B(K)=B(K)-B(S)*A(K,S)/A(S,S)
      DO 45 J=1,S-1
      A(K,J)=A(K,J)-A(K,S)*A(S,J)/A(S,S)
45    CONTINUE
      S=S-1
      GO TO 5
50    CONTINUE
      DO 70 I=1,N
      SUM=B(I)
      DO 60 J=1,I-1
      SUM=SUM-A(I,J)*X(J)
60    CONTINUE
      IF(ABS(A(I,I)).GT.0.0005)GO TO 61
      GO TO 100
```

```
61      X(I)=SUM/A(I,I)
70      CONTINUE
100                     CONTINUE
        RETURN
        END
```

```
C                                   APPENDIX A-3
C
C     ****************************************************************
C
C       APPROXIMATE MARKOV CHAIN MODEL FOR TP=TW
C
C     ****************************************************************
C
      DIMENSION S(17,17),CM(16,16),TRANS(17,17)
      INTEGER COMB(16,16)
      DIMENSION Z(17),B(17)
      INTEGER D
      COMMON TRANS,B,Z,MINPM
C
C
      TYPE 1973
1973              FORMAT(1X,'NUMBER OF PROCESSORS',/)
      ACCEPT 1974,N
1974              FORMAT(I)
      TYPE 1975
1975              FORMAT(1X,'NUMBER OF MEMORIES',/)
      ACCEPT 1974,M
      XM=1.0*M
      MINPM=MIN0(N,M)
C
C
      DO 300 I=1,17
      S(I,1)=0
300               S(1,I)=0
      S(1,1)=1
C
      DO 400 I=1,16
      DO 400 J=1,I
400               S(I+1,J+1)=1.0*J*S(I,J+1)+S(I,J)
C
      DO  500 I=1,16
      CM(I,1)=I
      DO 500 J=2,I
500               CM(I,J)=CM(I,J-1)*1.0*(I-J+1)
C
      COMB(1,1)=1
      DO 20 K=2,16
      COMB(K,1)=K
      L=MIN0(K,16)
      DO 10 I=2,L
10    COMB(K,I)=COMB(K,I-1)*(K-I+1)/I
```

```
 20      CONTINUE
 C
 C
         DO 1000 I=1,MINPM
 C
         XI=1.0*I/XM
 C
         DO 700 J=0,I
 C       J IS THE NUMBER OF ACTIVE PC'S THAT MAKE A REQUEST TO THE I MP'S
 C       XPROB IS THE PROB THAT N-I+J PC MAKE A REQUEST TO THE I MP'S
 C
         IF(J.EQ.0)XPROB=(1.0-XI)**I
         XMULT=(1.0-XI)**(I-J)
         IF(I.EQ.J)XMULT=1.0
         IF(J.NE.0)XPROB=1.0*COMB(I,J)*(XI**J)*XMULT
         NN=N-I+J
         K1=MIN0(NN,I)
         I1=1
         IF(K1.EQ.0)I1=0
         DO 700 L1=I1,K1
         TEMP1=1
         IF(L1.EQ.0)GO TO 595
         X1=(1.0*I)**NN
         TEMP1=CM(I,L1)*S(NN+1,L1+1)/X1
 C       NBUSY=NO. OF BUSY MP'S DURING NEXT CYCLE
 595                 K2=MIN0(I-J,M-I)
         I2=1
         IF(K2.EQ.0)I2=0
         DO 700 L2=I2,K2
         TEMP2=1
         NBUSY=L1+L2
         IF(L2.EQ.0)GO TO 695
         X2=(1.0*(M-I))**(I-J)
         TEMP2=CM(M-I,L2)*S(I-J+1,L2+1)/X2
 695                 TRANS(NBUSY,I)=TRANS(NBUSY,I)+XPROB*TEMP1*TEMP2
 700                 CONTINUE
 C
 1000                CONTINUE
 C
         DO 8000 I=1,MINPM
         TRANS(I,I)=TRANS(I,I)-1.0
 8000                TRANS(MINPM,I)=1.0
 C
         B(MINPM)=1
 C
         CALL GAUSS
```

Appendix A-3
Approximate Markov Chain Model for tp=tw

```
C
      DO 2000 I=1,MINPM
      UER=UER+1.0*Z(I)*I
2000                CONTINUE
C
      TYPE 1,UER
1     FORMAT(1X,F10.6)
C
      STOP
      END
C
C
      SUBROUTINE GAUSS
C     THIS SUBROUTINE SOLVES SIMULTANEOUS LINEAR EQUATIONS
C                 A*X=B
C
      DIMENSION A(17,17),B(17),X(17)
      COMMON A,B,X,N
      INTEGER S
C
      S=N
5     CONTINUE
      IF(S-1) 50,50,10
10    CONTINUE
      IF(S.GT.2)GO TO 105
      D=A(1,1)*A(2,2)-A(1,2)*A(2,1)
      IF(ABS(D).GT.0.0005)GO TO 105
      TYPE 25
      GO TO 100
105                 DO 20 I=1,S
      M=S-I+1
      IF(ABS(A(M,S)).GT.0.0005)GO TO 30
20    CONTINUE
      TYPE 25
25    FORMAT(1X,'THE COEFFICIENT MATRIX IS SINGULAR',/)
      GO TO 100
30    CONTINUE
      IF(M.EQ.S)GO TO 40
      T=B(S)
      B(S)=B(M)
      B(M)=T
      DO 35 J=1,S
      T=A(S,J)
      A(S,J)=A(M,J)
      A(M,J)=T
35    CONTINUE
```

```
40      CONTINUE
        DO 45 I=1,S-1
        K=S-I
        IF(ABS(A(S,S)).GT.0.0005)GO TO 42
        GO TO 100
42      B(K)=B(K)-B(S)*A(K,S)/A(S,S)
        DO 45 J=1,S-1
        A(K,J)=A(K,J)-A(K,S)*A(S,J)/A(S,S)
45      CONTINUE
        S=S-1
        GO TO 5
50      CONTINUE
        DO 70 I=1,N
        SUM=B(I)
        DO 60 J=1,I-1
        SUM=SUM-A(I,J)*X(J)
60      CONTINUE
        IF(ABS(A(I,I)).GT.0.0005)GO TO 61
        GO TO 100
61      X(I)=SUM/A(I,I)
70      CONTINUE
100                 CONTINUE
        RETURN
        END
```

```
C
C                                   APPENDIX A-4
C
C     ****************************************************************************
C
C     APPROXIMATE MODEL FOR ARBITRARY P(I,J), TP=TW, M>=N.
C
C                                   P(I,J)=ALPHA     FOR I=J
C                                         =B         OTHERWISE
C
C     ****************************************************************************
      DIMENSION UER(21),ALPHA(21),PROB(16,16)
      DIMENSION QPR(16,16)
      DIMENSION RATE(16),Y(16,16)
      DATA ALPHA/0.0,.05,.1,.15,.2,.25,.3,.35,.4,.45,.5,.55,.6,
     1.65,.7,.75,.8,.85,.9,.95,1.0/
      TYPE 1973
1973              FORMAT(1X,'NUMBER OF PROCESSORS',/)
      ACCEPT 1974,NPC
1974              FORMAT(I)
      TYPE 1975
1975              FORMAT(1X,'NUMBER OF MEMORIES',/)
      ACCEPT 1974,NMP
      MINPM=MIN0(NMP,NPC)
      XM=1.*NMP
      XN=1.0*NPC
C
C     COMPUTE UER FOR VARIOUS VALUES OF ALPHA
C
      DO 9999 IJK=1,21
      A=ALPHA(IJK)
      B=(1.0-A)/(XM-1)
      DO 11 I=1,NPC
      DO 10 J=1,NMP
10    PROB(I,J)=B
11    PROB(I,I)=A
C
C     COMPUTE QUEUEING FREQUENCIES BASED ON ACCESS FREQS.
C
      DO 70 I=1,NPC
      DO 70 J=1,NMP
      Y(I,J)=1.0
      DO 50 L=1,NPC
      IF(L.EQ.I)GO TO 50
      Y(I,J)=Y(I,J)*(1.0-PROB(L,J))
50    CONTINUE
70    CONTINUE
```

```
      DO 60 I=1,NPC
      DO 60 J=1,NMP
      QPR(I,J)=0.0
      DO 65 L=1,NPC
      IF(L.EQ.I)GO TO 65
      QPR(I,J)=PROB(L,J)*(1-Y(L,J))+QPR(I,J)
65    CONTINUE
      QPR(I,J)=(1.0-Y(I,J))*(QPR(I,J)+1.0)
60    QPR(I,J)=PROB(I,J)*(Y(I,J)+QPR(I,J))
      DO 80 I=1,NPC
      TEMP=0.0
      DO 85 J=1,NMP
85    TEMP=TEMP+QPR(I,J)
      DO 80 J=1,NMP
      QPR(I,J)=QPR(I,J)/TEMP
80    CONTINUE
43    FORMAT(1X,'Q(',I2,',',I2,')=',F9.5)
C
C     COMPUTE UER BASED ON QUEUEING FREQUENCIES
C
      DO 355 J=1,NMP
      RATE(J)=1.0
      DO 35 I=1,NPC
35    RATE(J)=(1.0-QPR(I,J))*RATE(J)
355                RATE(J)=1.0-RATE(J)
      DO 36 J=1,NMP
36    UER(IJK)=UER(IJK)+RATE(J)
C
9998                CONTINUE
      XMAX=1.0*MINPM
C
C     PLOT RESULTS
C
      CALL PLOT(21,XMAX,UER)
      WRITE(19,332),NPC,NMP
332                FORMAT(///,1X,'NUMBER OF PROCESSORS =',2X,I2,/,
     11X,'NUMBER OF MEMORIES =',2X,I2,//)
      DO 200 I=1,21
200                WRITE(19,331),ALPHA(I),UER(I)
331                FORMAT(1X,'ALPHA=',F5.3,5X,'UER=',F8.5)
      STOP
      END
C
C     PLOTTING SUBROUTINE
C
      SUBROUTINE PLOT(LIMIT,XMAX,VALUE)
```

Appendix A-4
Approximate Model for Arbitrary Pij, tp=tw, m≥n

```
      DIMENSION VALUE(100)
      DIMENSION J(101)
      J(100)='.'
C     WRITE(19,2)
2     FORMAT(11X,'.....................................................
      1.....................................................')
      DO 10 I=1,LIMIT
      IX=100.0*VALUE(I)/XMAX+0.5
      ILAST=IX
      DO 5 K=1,IX
5     J(K)=' '
      J(IX+1)='*'
      J(1)='.'
      WRITE(19,11),(J(L),L=1,IX+1)
11    FORMAT(11X,101(A1))
      WRITE(19,59)
59    FORMAT(/)
10    CONTINUE
      RETURN
      END
```

```
C                                   APPENDIX A-5
C
C     ****************************************************************************
C
C       APPROXIMATE MARKOV CHAIN MODEL FOR TP=TW+TC
C
C     ****************************************************************************
C
      DIMENSION S(17,17),CM(16,16),TRANS(17,17)
      DIMENSION Z(17),B(17)
      COMMON TRANS,B,Z,NSTATE
C
C
      TYPE 1973
1973              FORMAT(1X,'NUMBER OF PROCESSORS',/)
      ACCEPT 1974,N
1974              FORMAT(I)
      TYPE 1975
1975              FORMAT(1X,'NUMBER OF MEMORIES',/)
      ACCEPT 1974,M
      XM=1.0*M
      K=MIN0(N,M)
      NSTATE=K+1
C
      IF(K.NE.N)GO TO 100
      TRANS(NSTATE,1)=1.0
      I1=1
100               CONTINUE
C
      DO 300 I=1,17
      S(I,1)=0
300               S(1,I)=0
      S(1,1)=1
C
      DO 400 I=1,16
      DO 400 J=1,I
400               S(I+1,J+1)=1.0*J*S(I,J+1)+S(I,J)
C
      DO  500 I=1,16
      CM(I,1)=I
      DO 500 J=2,I
500               CM(I,J)=CM(I,J-1)*1.0*(I-J+1)
C
      DO 1000 II=I1+1,NSTATE
C
C     II....INDEX TO THE STATE DURING CURRENT CYCLE
```

Appendix A-5
Approximate Markov Chain Model for tp=tw+tc

```
C
      I=II+N-K-1
C
C     I......NUMBER OF PROCESSORS MAKING NEW REQUEST IN CURRENT CYCLE
C
      X=XM**I
      KK=MIN0(I,M)
      DO 900 JJ=1,KK
C
C     JJ.....NUMBER OF BUSY MPS IN CURRENT CYCLE
C     J......NUMBER OF PCS MAKING NEW REQUEST DURING NEXT CYCLE
C     J1.....INDEX TO STATE AT NEXT CYCLE
C
      J=N-JJ
      J1=J-N+NSTATE
      TRANS(J1,II)=1.0*CM(M,JJ)*S(I+1,JJ+1)/X
900                 CONTINUE
C
1000                CONTINUE
C
      DO 8000 I=1,NSTATE
      TRANS(I,I)=TRANS(I,I)-1.0
8000                TRANS(NSTATE,I)=1.0
C
      B(NSTATE)=1
C
      CALL GAUSS
C
      DO 2000 II=I1+1,NSTATE
      I=II+N-K-1
      UER=UER+XM*(1.0-(1.0-1.0/XM)**I)*Z(II)
2000                CONTINUE
C
      TYPE 1,UER
1     FORMAT(1X,F10.6)
C
      STOP
      END
C
C
      SUBROUTINE GAUSS
C     THIS SUBROUTINE SOLVES SIMULTANEOUS LINEAR EQUATIONS
C                   A*X=B
C
      DIMENSION A(17,17),B(17),X(17)
      COMMON A,B,X,N
```

```
      INTEGER S
C
      S=N
5     CONTINUE
      IF(S-1) 50,50,10
10    CONTINUE
      IF(S.GT.2)GO TO 105
      D=A(1,1)*A(2,2)-A(1,2)*A(2,1)
      IF(ABS(D).GT.0.0005)GO TO 105
      TYPE 25
      GO TO 100
105                DO 20 I=1,S
      M=S-I+1
      IF(ABS(A(M,S)).GT.0.0005)GO TO 30
20    CONTINUE
      TYPE 25
25    FORMAT(1X,'THE COEFFICIENT MATRIX IS SINGULAR',/)
      GO TO 100
30    CONTINUE
      IF(M.EQ.S)GO TO 40
      T=B(S)
      B(S)=B(M)
      B(M)=T
      DO 35 J=1,S
      T=A(S,J)
      A(S,J)=A(M,J)
      A(M,J)=T
35    CONTINUE
40    CONTINUE
      DO 45 I=1,S-1
      K=S-I
      IF(ABS(A(S,S)).GT.0.0005)GO TO 42
      GO TO 100
42    B(K)=B(K)-B(S)*A(K,S)/A(S,S)
      DO 45 J=1,S-1
      A(K,J)=A(K,J)-A(K,S)*A(S,J)/A(S,S)
45    CONTINUE
      S=S-1
      GO TO 5
50    CONTINUE
      DO 70 I=1,N
      SUM=B(I)
      DO 60 J=1,I-1
      SUM=SUM-A(I,J)*X(J)
60    CONTINUE
      IF(ABS(A(I,I)).GT.0.0005)GO TO 61
```

Appendix A-5
Approximate Markov Chain Model for tp=tw+tc

```
      GO TO 100
61    X(I)=SUM/A(I,I)
70    CONTINUE
100                 CONTINUE
      RETURN
      END
```

```
C                                   APPENDIX A-6
C
C     ***********************************************************************
C
C       APPROXIMATE MARKOV CHAIN MODEL FOR TP>TW
C     PROB(TP=TW+I*TC)=BETA*ALPHA**I
C                     E(TP)=TW+TC*ALPHA/BETA
C                     ALPHA+BETA=1
C
C     ***********************************************************************
C
      DIMENSION S(17,17),CM(16,16),TRANS(17,17)
      INTEGER COMB(16,16)
      DIMENSION Z(17),B(17)
      INTEGER D
      COMMON TRANS,B,Z,NSTATE
C
C
      TYPE 1973
1973                FORMAT(1X,'NUMBER OF PROCESSORS',/)
      ACCEPT 1974,N
1974                FORMAT(I)
      TYPE 1975
1975                FORMAT(1X,'NUMBER OF MEMORIES',/)
      ACCEPT 1974,M
      XM=1.0*M
      TYPE 1976
1976                FORMAT(1X,'ALPHA : PROB OF ONE MORE CYCLE OF
     EXECUTION',/)
      ACCEPT 1977,ALPHA
1977                FORMAT(F)
      BETA=1.0-ALPHA
      NSTATE=N+1
C
C
      DO 300 I=1,17
      S(I,1)=0
300                 S(1,I)=0
      S(1,1)=1
C
      DO 400 I=1,16
      DO 400 J=1,I
400                 S(I+1,J+1)=1.0*J*S(I,J+1)+S(I,J)
C
      DO  500 I=1,16
      CM(I,1)=I
```

Appendix A-6
Approximate Markov Chain Model for Geometric tp

```
      DO 500 J=2,I
500               CM(I,J)=CM(I,J-1)*1.0*(I-J+1)
C
      COMB(1,1)=1
      DO 20 K=2,16
      COMB(K,1)=K
      L=MIN0(K,16)
      DO 10 I=2,L
10    COMB(K,I)=COMB(K,I-1)*(K-I+1)/I
20    CONTINUE
C
C
      DO 1000 I=1,N
      X=XM**I
C
C     K IS THE MAX. NO. OF OCCUPIED MP'S
C     D IS THE NUMBER OF OCC. MP'S
C
      K=MIN0(I,M)
      DO 1000 D=1,K
C
C     I-D  PC ARE LEFT IN MP QUEUES
C     NN  PC ARE TO BE REASSIGNED TO PC OR MP QUEUES
C     XPROB IS THE PROB OF D MP'S BEING BUSY
      XPROB=CM(M,D)*S(I+1,D+1)/X
900               NN=N-I+D
      TRANS(I-D+1,I+1)=TRANS(I-D+1,I+1)+XPROB*ALPHA**NN
      IF(NN.EQ.0)GO TO 1000
C
      DO 1000 NEWPC=1,NN
C
C     NEWPC= NO. OF NEW PC'S TO BE ASSIGNED TO MP QUEUES
C     NEWMP= TOTAL NO. OF PC'S THAT MAKE MP REQ. NEXT CYCLE
C
      NEWMP=I-D+NEWPC
C
      TEMP=1.0*COMB(NN,NEWPC)*(BETA**NEWPC)*(ALPHA**(NN-NEWPC))
      TRANS(NEWMP+1,I+1)=TRANS(NEWMP+1,I+1)+XPROB*TEMP
C
1000              CONTINUE
C
      DO 1500 I=1,NSTATE
1500              TRANS(I,1)=TRANS(I,2)
      DO 8000 I=1,NSTATE
      TRANS(I,I)=TRANS(I,I)-1.0
8000              TRANS(NSTATE,I)=1.0
```

```
C
      B(NSTATE)=1
C
      CALL GAUSS
C
      DO 2000 I=1,N
      UER=UER+XM*(1.0-(1.0-1.0/XM)**I)*Z(I+1)
2000              CONTINUE
C
      TYPE 1,UER
1     FORMAT(1X,F10.6)
C
      STOP
      END
C
C
      SUBROUTINE GAUSS
C     THIS SUBROUTINE SOLVES SIMULTANEOUS LINEAR EQUATIONS
C                 A*X=B
C
      DIMENSION A(17,17),B(17),X(17)
      COMMON A,B,X,N
      INTEGER S
C
      S=N
5     CONTINUE
      IF(S-1) 50,50,10
10    CONTINUE
      IF(S.GT.2)GO TO 105
      D=A(1,1)*A(2,2)-A(1,2)*A(2,1)
      IF(ABS(D).GT.0.0005)GO TO 105
      TYPE 25
      GO TO 100
105               DO 20 I=1,S
      M=S-I+1
      IF(ABS(A(M,S)).GT.0.0005)GO TO 30
20    CONTINUE
      TYPE 25
25    FORMAT(1X,'THE COEFFICIENT MATRIX IS SINGULAR',/)
      GO TO 100
30    CONTINUE
      IF(M.EQ.S)GO TO 40
      T=B(S)
      B(S)=B(M)
      B(M)=T
      DO 35 J=1,S
```

Appendix A-6
Approximate Markov Chain Model for Geometric tp

```
      T=A(S,J)
      A(S,J)=A(M,J)
      A(M,J)=T
35    CONTINUE
40    CONTINUE
      DO 45 I=1,S-1
      K=S-I
      IF(ABS(A(S,S)).GT.0.0005)GO TO 42
      GO TO 100
42    B(K)=B(K)-B(S)*A(K,S)/A(S,S)
      DO 45 J=1,S-1
      A(K,J)=A(K,J)-A(K,S)*A(S,J)/A(S,S)
45    CONTINUE
      S=S-1
      GO TO 5
50    CONTINUE
      DO 70 I=1,N
      SUM=B(I)
      DO 60 J=1,I-1
      SUM=SUM-A(I,J)*X(J)
60    CONTINUE
      IF(ABS(A(I,I)).GT.0.0005)GO TO 61
      GO TO 100
61    X(I)=SUM/A(I,I)
70    CONTINUE
100                 CONTINUE
      RETURN
      END
```

```
C       ********************************************************************
C
C       CACHE MODEL FOR TP>=TW, CONSTANT.
C                                       APPENDIX A-7
C
C
C       ********************************************************************
        TYPE 1
        ACCEPT 4,TP
1       FORMAT(1X,'ENTER VALUE OF TP,FORMAT F',/)
        TYPE 2
        ACCEPT 4,TF
2       FORMAT(1X,'ENTER VALUE OF TF',/)
        TYPE 5
        ACCEPT 4,TC
5       FORMAT(1X,'ENTER VALUE OF TC',/)
        TYPE 6
        ACCEPT 4,TW
6       FORMAT(1X,'ENTER VALUE OF TW',/)
4       FORMAT(F)
        TYPE 1973
1973                FORMAT(1X,'NUMBER OF PROCESSORS, INTEGER FORMAT',/)
        ACCEPT 1974,N
1974                FORMAT(I)
        TYPE 1975
1975                FORMAT(1X,'NUMBER OF MEMORIES',/)
        ACCEPT 1974,M
        XN=1.0*N
        XM=1.0*M
        TA=TC-TW
        DO 2000 I=0,9
        ALPHA=0.1*I
        BETA=1.0-ALPHA
        CONS=ALPHA*(TP+TF)+BETA*(TP-TW)
C
        PMAX=1.0
        PMIN=0.0
        P=0.5
10      PNEW=1.0-CONS*(1.0-(1.0-P/XM)**N)*XM/XN/TC/BETA
        DEL=P-PNEW
        DEL=ABS(DEL)
        IF(DEL.LE.0.0001)GO TO 1000
        IF(PNEW.GT.P)PMIN=P
        IF(PNEW.LT.P)PMAX=P
        P=0.5*(PMIN+PMAX)
        GO TO 10
```

Appendix A-7

Cache Model for tp≥tw

```
C
1000                    P=0.5*(PNEW+P)
      UER=1.0-(1.0-P/XM)**N
      UER=UER*XM/BETA
2000                    TYPE 20,UER
20    FORMAT(1X,'EXECUTION RATE=',F8.4)
      STOP
      END
```

APPENDIX B

MULTIPROCESSOR SYSTEM SIMULATOR

This simulator can be used for multiprocessor systems with NPC processors and NMP memory units connected by a single crosspoint switch. Pc[i] has a probability PROB(I,J) of requesting service from memory j. The matrix PROB can be carefully chosen to model partially assigned crosspoints and private caches e.g. if memory unit 1 is a private cache dedicated to Pc[1], then PROB(1,1)= Prob{Pc[i] hits cache] and PROB(K,1)=0 for K≠1.

The simulation program consists of four main parts:

(i) Main Program

(ii) Subroutine PC(ID)

(iii)Subroutine MP(ID)

(iv) BLOCK DATA

The main program contains the scheduler, which calls the two subroutines when a processor or memory is activated. This is a discrete event simulator where the two subroutines are analogous to activities in SIMULA. The scheduling is done by maintaining a doubly linked list called the sequencing set or SQS. Figure B-1 shows the structure of the SQS array.

Successor	Predecessor	Scheduled Activation Time	Event Type	Event Identification Number

Figure B-1 Structure of the SQS array

The first element or the head of the SQS is a dummy element, whose successor is the first real event notice. Thus, when the SQS is empty (i.e. no event notices) the first element of the SQS points to itself. The SQS is maintained in proper time order with the next most imminent event as the successor of the head.

Figure B-2 depicts a simple flow chart for the main program. The program executes in a loop until a preset simulation time is reached. At this stage it jumps out of the loop and operates on the statistics collected and outputs parameters like the number of instructions executed, the execution rate, and the queue lengths and waiting times for the various memories. A typical output is shown in fig. B-3.

The processor activity is characterized by a subroutine PC(ID); the flow chart is depicted in fig B-4. A uniform random number with range (0,1) is compared with the access probabilities in the array PROB to select a memory. A request is entered into the queue corresponding to that memory unit. An event notice for activating the memory is entered into the SQS if necessary.

Figure B-5 illustrates the working of the subroutine MP(ID). In its current version, a processor is selected as per FIFO discipline. The processor is then scheduled to be activated after a time interval equal to the sum of the memory access time(ta) and the processor's execution time(tp). The processing time distribution can be arbitrarily chosen. The program listed here has an

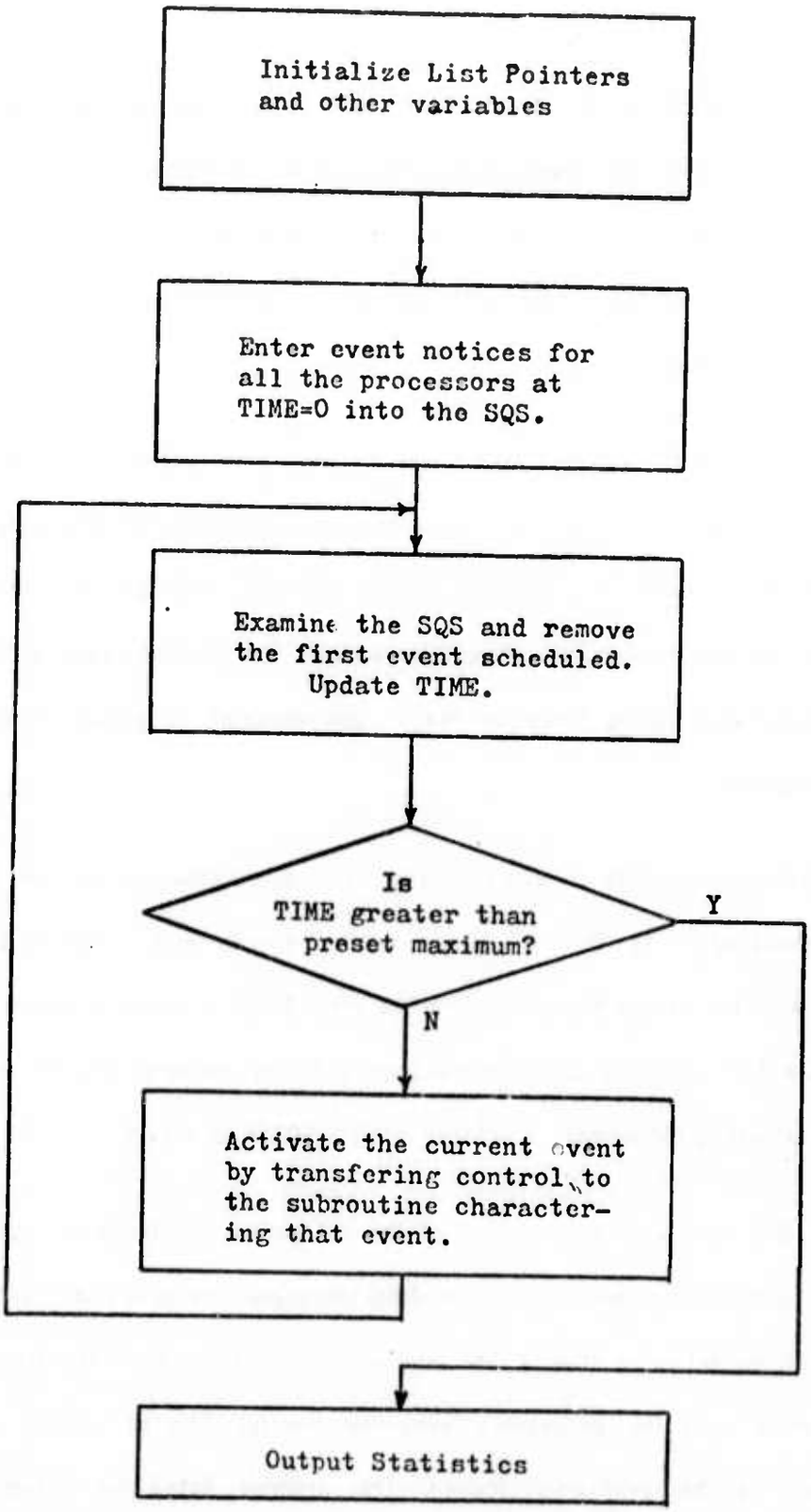


Figure B-2 Flow chart of the main program.

```
NO. OF UNIT INSTR EXEC IN        500000  TIMEUNITS IS        3465
UNIT EXEC RATE= 0.006930

UNIT INSTRUCTIONS EXECUTED BY INDIVIDUAL PROCESSORS
       868
       869
       864
       864

ACCESS FREQUENCIES

0.297235  0.381336  0.321429
0.303797  0.331415  0.364787
0.303241  0.365741  0.331019
0.312500  0.343750  0.343750

******* MEMORY UNIT  1 *******
NO. OF REFS=         1054
MAX Q LENGTH= 2
AVG. Q LENGTH= 0.01467
AVG. WAITING TIME=    6.9573

******* MEMORY UNIT  2 *******
NO. OF REFS=         1232
MAX Q LENGTH= 2
AVG. Q LENGTH= 0.02154
AVG. WAITING TIME=    8.7419

******* MEMORY UNIT  3 *******
NO. OF REFS=         1179
MAX Q LENGTH= 2
AVG. Q LENGTH= 0.02107
AVG. WAITING TIME=    8.9372

AVERAGE VALUE=   0.02108

AVERAGE VALUE=   0.02464

AVERAGE VALUE=   0.02358
```

Figure B-3 Typical Simulator Output

exponentially distributed processing time. The memory reschedules itself at the end of the cycle if any request are pending in its queue.

The BLOCK DATA contains input parameters for the simulation.

This simulator could have been written in SIMULA. However, the scheduling involved does not necessitate all the capabilities of SIMULA. Moreover, the easy access to FORTRAN on the PDP-10's time-sharing system was an important consideration in its selection.

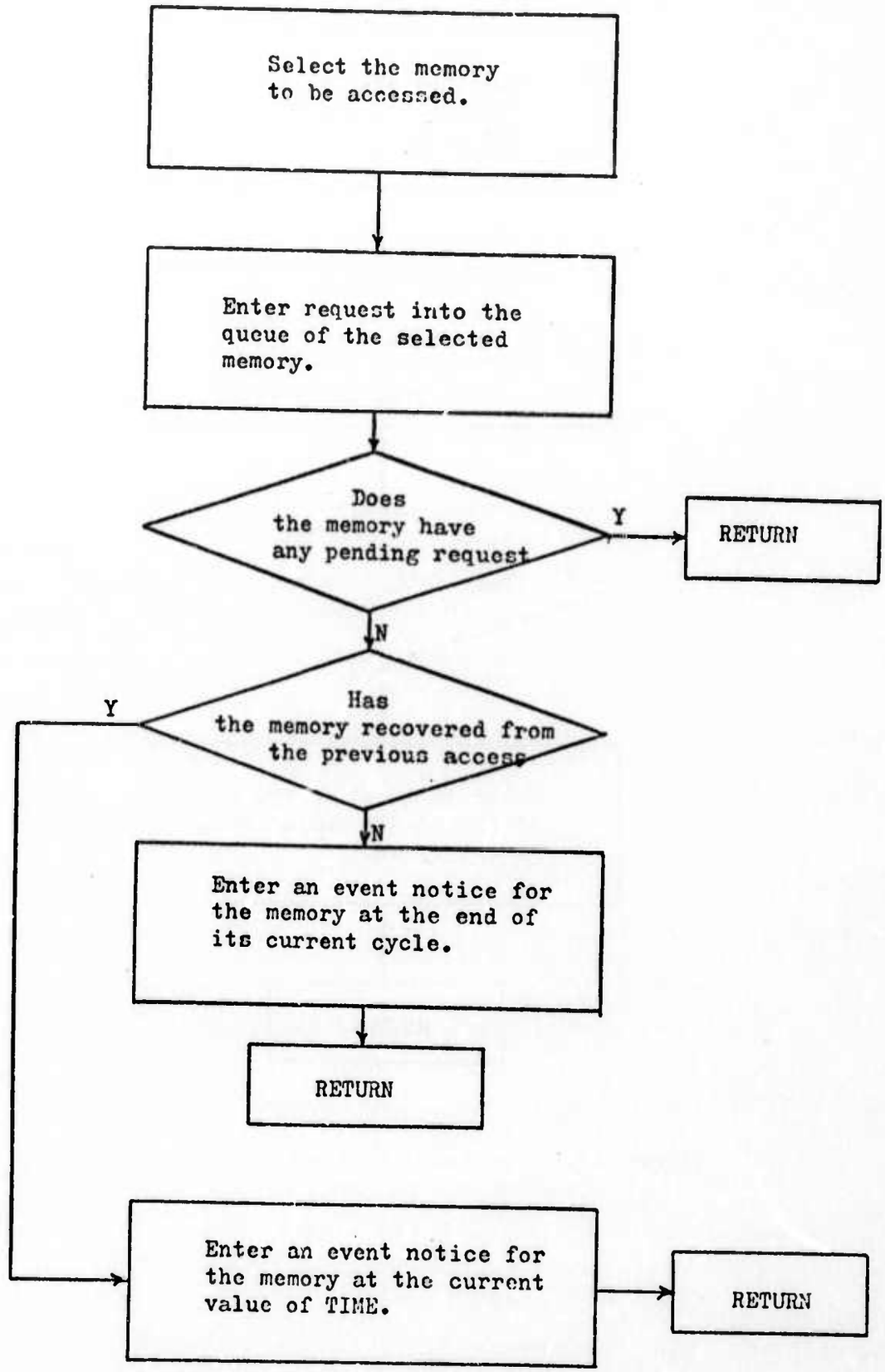


Figure B-4 Flow chart of the processor's activity.

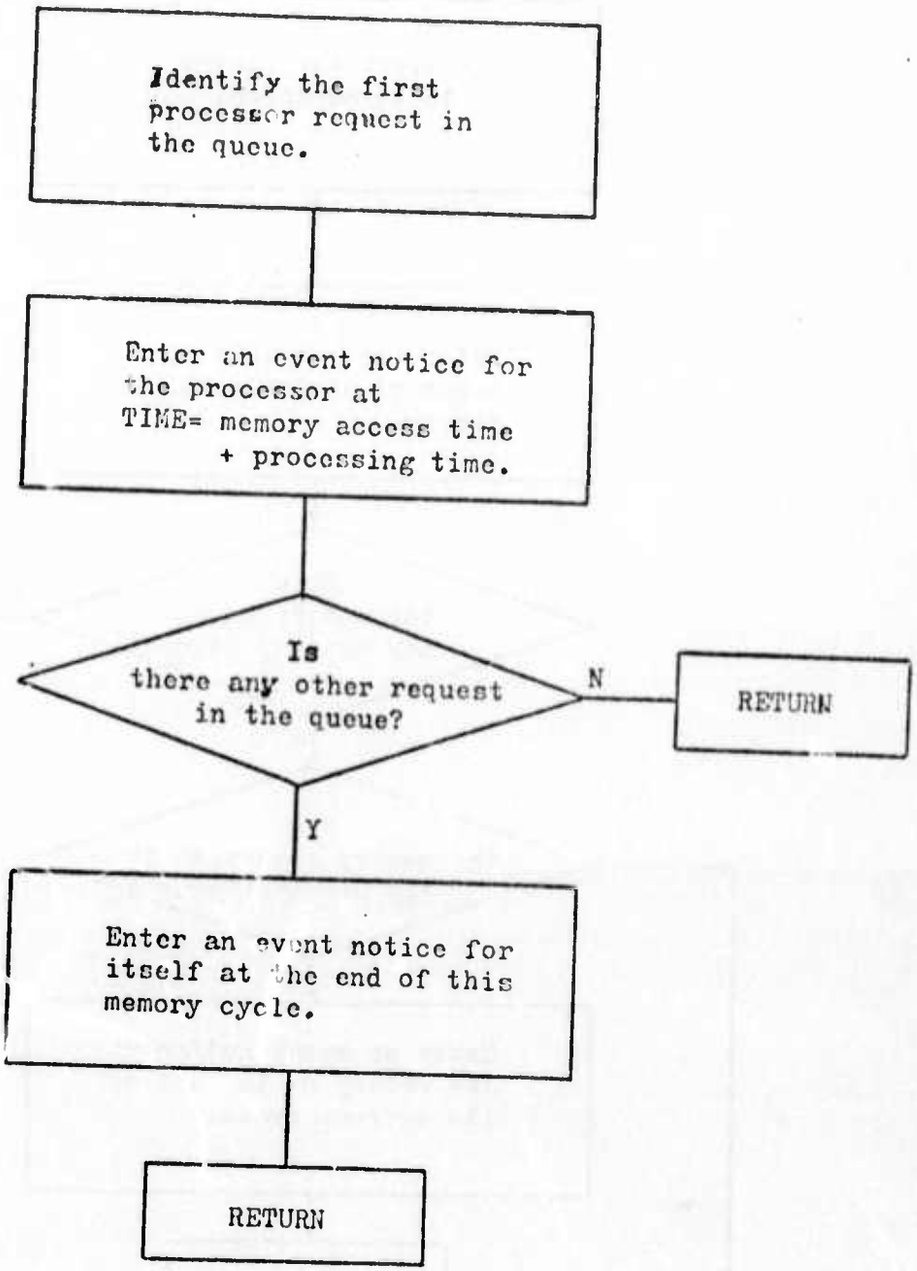


Figure B-5 Flow chart of the memory's activity.

```
C       PROCESSING TIME --- EXPONENTIAL
C       THIS PROGRAM SIMULATES A MULTIPROCESSOR WITH A
C       CROSS-POINT BETWEEN PROCESSORS & MEMORIES
        INTEGER SQS(100.5),EMPTY(99)
        INTEGER SUC,PREO,SCHED,EVTYPE,EVID,EVNUM,TOPMT
        COMMON/SQSET/ SQS,EMPTY,SUC,PREO,SCHEO,EVTYPE,EVID,TOPMT
        INTEGER TA(50),TC(50),TP(50)
        INTEGER TIME,SIMTIM
        COMMON/GLOBAL/ TA,TC,TP,TIME,NPC,NMP,SIMTIM,NINSTR
        DIMENSION COUNT(50,50),INSTR(50)
        COMMON/MEM/COUNT,INSTR
        DIMENSION PROB(50,50)
        COMMON/PROC/ PROB,ISEED
        INTEGER TTQ(50),TLAST(50),LENGTH(50),MAXQL(50)
        COMMON/STAT/TTQ,TLAST,LENGTH,MAXQL
        INTEGER QMP(50,50),FIRST(50),NEXT(50),NEXTAC(50)
        COMMON/Q/ QMP,FIRST,NEXT,NEXTAC
        COMMON/EXP/ISD
C       INITIALIZE SQS AND EMPTY LIST
        DIMENSION RATE(50),JMP(50),JPC(50)
        DIMENSION QPR(50,50),TEMP(50)
        DATA SUC,PREO,SCHED,EVTYPE,EVID/1,2,3,4,5/
        DATA TOPMT/1/
        DO 21 I=1,99
21      EMPTY(I)=I+1
        DO 22 I=1,NMP
        NEXT(I)=1
22      FIRST(I)=1
        SQS(1,SUC)=1
        SQS(1,PRED)=1
C       ACTIVATE ALL PROCESSORS AT TIME=0
        DO 23 I=1,NPC
23      CALL  INSERT(1,I,0)
C       CONVERT ACCESS PROBS. TO CUMULATIVE ACCESS PROBS.
        DO 24 I=1,NPC
        DO 24 J=2,NMP
24      PROB(I,J)=PROB(I,J)+PROB(I,J-1)
C       THIS IS THE SCHEDULER
C       REMOVE  'FIRST' ELEMENT IN SQS AND RESTORE PTRS.
1000    I=SQS(1,SUC)
        IF(I.EQ.1)TYPE 701
701     FORMAT(1X,'EVENT LIST EMPTY')
        J=SQS(I,SUC)
        SQS(1,SUC)=J
        SQS(J,PREO)=1
C       UPDATE TIME
        TIME=SQS(I,SCHEO)
        IF(TIME.GE.SIMTIM)GO TO 2001
        EVNUM=SQS(I,EVTYPE)
        ID=SQS(I,EVIO)
C       UPDATE EMPTY LIST
        TOPMT=TOPMT-1
        EMPTY(TOPMT)=I
C       ACTIVATE CURRENT EVENT
        GO TO (1,2) EVNUM
```

```
        TYPE 702
702     FORMAT(1X,'SCHEDULER HAS UNKNOWN EVENT TYPE')
        STOP
1       CALL PC(ID)
        GO TO 1000
2       CALL MP(ID)
        GO TO 1000
C       OUTPUT STATISTICS
2001    CONTINUE
        DO 2002 I=1,NMP
2002    NINSTR=NINSTR+INSTR(I)
        TIME=SIMTIM
        WRITE(19,41),TIME,NINSTR
41      FORMAT(1X,'NO. OF UNIT INSTR EXEC IN',2X,I12,2X,'TIME
        1UNITS IS',2X,I12)
        UER=NINSTR/TIME*1.0
        WRITE(19,40),UER
        TYPE 40,UER
40      FORMAT(1X,'UNIT EXEC RATE=',F10.6)
C       FIND NO. OF INSTR EXEC BY EACH PROCESSOR
        DO 31 I=1,NPC
        DO 31 J=1,NMP
31      JPC(I)=JPC(I)+COUNT(I,J)
C       FIND NO. OF ACCESSES TO EACH MEMORY
        DO 32 I=1,NMP
        DO 32  J=1,NPC
32      JMP(I)=JMP(I)+COUNT(J,I)
        WRITE(19,42),(JPC(I),I=1,NPC)
42      FORMAT(/,1X,'UNIT INSTRUCTIONS EXECUTED BY INDIVIDUAL
        1 PROCESSORS',/,50(1X,I10,/))
C       FIND ACCESS FREQUENCIES FOR THIS SIMULATION RUN
        DO 33 I=1,NPC
        DO 33 J=1,NMP
33      PROB(I,J)=COUNT(I,J)/JPC(I)
        WRITE(19,443)
443     FORMAT(1X,'ACCESS FREQUENCIES',/)
        DO 333 I=1,NPC
333     WRITE(19,43),(PROB(I,J),J=1,NMP)
43      FORMAT(1X,8(F8.6,2X))
C       COMPUTE AVG WAIT TIME,AVG Q LENGTH
        DO 34 I=1,NMP
        AVQL=1.0*TTQ(I)/TIME
        AVWT=1.0*TTQ(I)/JMP(I)
34      WRITE(19,44),I,JMP(I),MAXQL(I),AVQL,AVWT
44      FORMAT(/,1X,'******* MEMORY UNIT ',I2,' *******',/,
        11X,'NO. OF REFS=',I12,/,1X,'MAX Q LENGTH=',I2,/,
        11X,'AVG. Q LENGTH=',F8.5,/,1X,'AVG.
        2WAITING TIME=',F10.4)
        N=TIME/10
        DO 801 ID=1,NMP
        AV=INSTR(ID)/TIME*10.0
801     WRITE(19,202),AV
202     FORMAT(/,1X,'AVERAGE VALUE=',F10.5)
        STOP
        END
```

```
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        SUBROUTINE PC(ID)
        DIMENSION PROB(50,50)
        COMMON/PROC/ PROB,ISEED
        INTEGER QMP(50,50),FIRST(50),NEXT(50),NEXTAC(50)
        COMMON/Q/ QMP,FIRST,NEXT,NEXTAC
        INTEGER TA(50),TC(50),TP(50)
        INTEGER TIME,SIMTIM
        COMMON/GLOBAL/ TA,TC,TP,TIME,NPC,NMP,SIMTIM,NINSTR
        INTEGER TTQ(50),TLAST(50),LENGTH(50),MAXQL(50)
        COMMON/STAT/TTQ,TLAST,LENGTH,MAXQL
C       PROB(I,J)= PROBABILITY(PC(I) ACCESSES PC(J))
C       GENERATE A UNIFORM RV & DETERMINE MEMORY TO BE ACCESSED
1       RV=UNIRAN(ISEED)
        RV=RV+1.0/NPC*IO
        IF(RV.GT.1.0)RV=RV-1
        DO 10 I=1,NMP
        IF(RV.LE.PROB(ID,I))GO TO 100
10      CONTINUE
        GO TO 1
100     IMP=I
C       INCREMENT Q MEASURE COUNTERS
        TTQ(IMP)=TTQ(IMP)+(TIME-TLAST(IMP))*LENGTH(IMP)
        LENGTH(IMP)=LENGTH(IMP)+1
        IF(LENGTH(IMP).GT.MAXQL(IMP))MAXQL(IMP)=LENGTH(IMP)
        TLAST(IMP)=TIME
C       PUT IN REQUEST IN QUEUE FOR CHOSEN MEMORY
        QMP(IMP,NEXT(IMP))=ID
        NEXT(IMP)=NEXT(IMP)+1
        IF(NEXT(IMP).GT.NPC)NEXT(IMP)=1
C       CHECK IF MEMORY UNIT HAS OTHER REQUESTS QUEUED
        IF(LENGTH(IMP).GT.1)RETURN
C       CHECK IF MEMORY IS READY TO GRANT THIS ACCESS NOW
        IF(TIME.GT.NEXTAC(IMP))GO TO 200
        CALL INSERT(2,IMP,NEXTAC(IMP))
        RETURN
200     CALL INSERT(2,IMP,TIME)
        RETURN
        END
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

        SUBROUTINE MP(ID)
        INTEGER QMP(50,50),FIRST(50),NEXT(50),NEXTAC(50)
        COMMON/Q/ QMP,FIRST,NEXT,NEXTAC
        INTEGER TA(50),TC(50),TP(50)
        INTEGER TIME,SIMTIM
        COMMON/GLOBAL/ TA,TC,TP,TIME,NPC,NMP,SIMTIM,NINSTR
        DIMENSION COUNT(50,50),INSTR(50)
        COMMON/MEM/COUNT,INSTR
        INTEGER TTQ(50),TLAST(50),LENGTH(50),MAXQL(50)
        COMMON/STAT/TTQ,TLAST,LENGTH,MAXQL
        COMMON/EXP/ISEED
        IF(LENGTH(ID).EQ.0)TYPE 703
703     FORMAT(1X,'MEMORY ACTIVATED WITHOUT SCHEDULED REQUEST')
C       INCREMENT Q MEASURE COUNTERS
```

```
      INSTR(ID)=INSTR(ID)+1
      TTQ(ID)=TTQ(ID)+(TIME-TLAST(ID))*LENGTH(ID)
      LENGTH(ID)=LENGTH(ID)-1
      TLAST(ID)=TIME
C     IDENTIFY PC TO BE SERVICED AS PER FIFO STRATEGY
      IPC=QMP(ID,FIRST(ID))
      FIRST(ID)=FIRST(ID)+1
      COUNT(IPC,ID)=COUNT(IPC,ID)+1
      IF(FIRST(ID).GT.NPC)FIRST(ID)=1
9374  RV=UNIRAN(ISEED)
      TPROC=450-TP(IPC)*ALOG(RV)
      IF(TPROC.GT.10*TP(IPC))GO TO 9374
      JTIME=TIME+TA(ID)+TPROC
C     SCHEDULE PC
      CALL INSERT(1,IPC,JTIME)
      NEXTAC(ID)=TIME+TC(ID)
      IF(LENGTH(ID).EQ.0)RETURN
      CALL INSERT(2,ID,NEXTAC(ID))
      RETURN
      END

C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

      SUBROUTINE INSERT(JTYPE,ID,TSCHED)
C     THIS ROUTINE INSERTS ELEMENT I AFTER K WRT TIME
      INTEGER TSCHED
      INTEGER SQS(100,5),EMPTY(99)
      INTEGER SUC,PRED,SCHED,EVTYPE,EVID,EVNUM,TOPMT
      COMMON/SQSET/ SQS,EMPTY,SUC,PRED,SCHED,EVTYPE,EVID,TOPMT
300   IF(TOPMT.EQ.0) TYPE 704
704   FORMAT(1X,'SCHEDULER OVERFLOW')
      IF(SQS(1,SUC).EQ.1) GO TO 320
      J=SQS(1,SUC)
310   CONTINUE
      IF(SQS(J,SCHED).GT.TSCHED) GO TO 330
      IF(SQS(J,SUC).EQ.1)GO TO 340
      J=SQS(J,SUC)
      GO TO 310
320   K=1
      GO TO 100
330   K=SQS(J,PRED)
      GO TO 100
340   K=J
100   I=EMPTY(TOPMT)
      SQS(I,SUC)=SQS(K,SUC)
      J=SQS(K,SUC)
      SQS(I,PRED)=K
      SQS(K,SUC)=I
      SQS(J,PRED)=I
      SQS(I,SCHED)=TSCHED
      SQS(I,EVTYPE)=JTYPE
      SQS(I,EVID)=ID
      TOPMT=TOPMT+1
      RETURN
      END
```

```
      REAL FUNCTION UNIRAN(JSEED)
      JSEED=JSEED*3141592621
      JSEED=JSEED+7261067113
      IF(JSEED)1,2,2
1     JSEED=JSEED+34359738367+1
2     TEMP=JSEED
      UNIRAN=TEMP*0.2910380346E-10
      RETURN
      END
```